Applications of Model Checking in the Context of Cyber Security for Digital Twins

Douglas Fraser

Submitted in fulfilment of the requirements for the
Degree of Doctor of Philosophy

School of Computing Science
College of Science and Engineering
University of Glasgow

February 2025

Abstract

Cyber security attacks on Industrial Control Systems (ICSs) are increasingly sophisticated, targeting their ability to manage critical processes and posing risks to national infrastructure. Addressing this threat requires innovative methods to ensure the secure design and operation of ICS. Digital Twins (DTs) have emerged as a promising tool for enhancing the efficiency and cyber security of the systems they represent; however, their effectiveness depends on reliable intrusion detection methods and secure integration within existing industrial control environments. Securely deploying a DT to an ICS requires careful consideration of existing architecture and the potential security risks of incorporating the DT itself. Formal methods, in particular model checking, are an effective tool for analysing system design and detecting cyber security vulnerabilities.

We present two complementary applications of model checking techniques to support the deployment of DTs in ICS environments. We first develop a specification-based intrusion detection approach utilising the SPIN model checker and deploy it into a DT environment for a hydroelectric dam testbed. We explain the process we followed to develop Promela models from PLC code to detect inconsistencies between received data and specified system behaviours. Our evaluation shows that the models achieved performance on a par with machine learning approaches while maintaining explainability and delivering metrics of 99.99% precision, 99.05% recall, a 99.52% F1-score, and 99.05% accuracy.

We then address the expanded attack surface that can result from integrating DTs into ICSs. We explore this issue by developing a series of Alloy models that consider the dataflow between a DT and its underlying asset. The developed models incorporate novel modelling of an attacker's action space to represent how threat actors can move through a network. Using our approach, we model our hydroelectric testbed DT to identify security vulnerabilities in our design and develop an improved network design to mitigate them. Our approach successfully identified security vulnerabilities within the DT-ICS integration and informed network design improvements to reduce the attack surface significantly. Our evaluation confirms that model checking techniques enhance both intrusion detection and security assessment, offering a structured and explainable alternative to machine learning methods. We discuss the merits and drawbacks of each of our approaches and discuss methods of expanding and improving them to support DT development.

Contents

List of Tables

List of Figures

Acknowledgements

This thesis would not have been possible without the support and guidance of many people to whom I owe my thanks.

Firstly, thank you to Prof. Alice Miller for the opportunity to work on this project, her continual guidance throughout my Ph.D. and for her patience while I explored the many, many tangents that eventually led me to the work enclosed in this thesis. I would also like to thank Marco Cook of the University of Glasgow ICS Lab, who has been a great mentor to me throughout my Ph.D. journey and who trusted me with all of his shiny ICS equipment.

My research was supported by a number of very talented and generous people in the Dstl Cyber Defence lab that I was fortunate enough to have lend me their time, expertise and, most importantly, their good sense of fun. Thank you to Andy, Colin, and Matt for their technical guidance in this project and the many laughs along the way. I'd especially like to thank Sarah, who managed the project and gave me many more excellent opportunities along the way, and who brings such a genuine warmth and dedication to her work.

This thesis was made possible by my excellent friends, who supported and inspired me through this process. Thank you to the regulars of 5 Ruthven Street, who are always keen for a new adventure and always looking for ways to turn things into games. A special mention goes to Iain and Dobbs, who not only proofread this thesis as promised but did so with great care and a welcome sense of humour. Lastly, thank you to Tree, Leah and Ross, with whom I've had the joy of sharing a home these last four years. Though we've not started a commune yet, I think in our flat we've found the next best thing.

The biggest thanks, however, go to my family. To my brother, Euan, thank you for continuing to put up with me and for a lot of great memories running around, hitting each other with wooden swords. Though we now live on opposite sides of the country and were forced to hang up our swords a long time ago, it is a gift to have such an inspiring brother, and whatever time we have together is time treasured. Finally, I've been very blessed in my life to have two wonderful parents, Ken and Maureen, who have given me so many of the tools and qualities that got me to where I am. Still, the greatest gift has been their enduring love and support throughout. This Ph.D. was considerably longer than any swimming competition has ever been (though thankfully involved less waiting around at swimming pools), but you've always been there, feeding me snacks, cheering me on and reminding me that when you see the flags, you get your head down and go for it. For that and so much more, thank you.

Declaration

All work in this thesis was carried out by the author.

Acronyms

AI Artificial Intelligence.

API Application Programming Interface.

BDD Binary Decision Diagram.

CDT Cyber Security Digital Twin.

CPS Cyber-Physical System.

CTL Computation Tree Logic.

CTMC Continuous-time Markov Chain.

DMZ Demilitarised Zone.

DoS Denial of Service.

DT Digital Twin.

DTMC Discrete-time Markov Chain.

HMI Human Machine Interface.

ICS Industrial Control System.

IIoT Industrial Internet of Things.

IoT Internet of Things.

IP Internet Protocol.

IT Information Technology.

LTL Linear Temporal Logic.

MDP Markov Decision Process.

ML Machine Learning.

OT Operational Technology.

PLC Programmable Logic Controller.

PLM Product Lifecycle Management.

RTU Remote Terminal Unit.

SCADA Supervisory Control and Data Acquisition.

TCP Transmission Control Protocol.

TCTL Timed Computation Tree Logic.

UDP User Datagram Protocol.

Chapter 1

Introduction

1.1 Motivation

Industrial Control Systems (ICSs) are a critical part of modern society. They maintain the water that we drink, provide the power to our homes and ensure the safe operation of transportation networks, manufacturing processes, and essential services such as healthcare and telecommunications. These systems play a fundamental role in sustaining daily life, economic stability, and national security. However, as ICSs become more interconnected and digitised, they face a growing number of cyber threats that can disrupt operations, compromise safety, and lead to significant financial and reputational damage. Faced with increasingly sophisticated attacks targeting these critical systems, their safeguarding requires the development of innovative measures that can detect and mitigate potential intrusions without compromising their performance and reliability.

ICSs are used in environments that require high availability and reliability in order to manage the needs of the processes that they control and to maintain a safe working environment. ICS technology tends to evolve at a slower rate than in other areas of computing, and the adoption of new concepts is slower so that new technologies are proven to be dependable. The term Industrial Internet of Things (IIoT) refers to the recent increase in integrating smart sensors and actuators into traditional ICS processes. This has led to ICSs becoming more heterogeneous and has been accompanied by increased digitisation and interconnection between systems. This accessibility of Operational Technology (OT) systems facilitates the mass collection and dissemination of system data, enabling operators to monitor performance. Many end control devices were designed to prioritise performance and safety in an era when they operated in an isolated, "air-gapped" network that was completely disconnected from the outside world. As a result, these systems and the protocols they utilise to communicate are easily manipulated by adversaries who can access them. Connecting the OT environment to an enterprise network dramatically increases the attack surface that unauthorised users can use to access these vulnerable systems. Reducing the threat to these systems requires the secure design and monitoring of the

OT network.

One technique that can be used to detect attacks is anomaly detection. This is the process of identifying device behaviours inconsistent with the system's regular operation. These behaviours can occur either through a software bug, hardware fault or through the malicious intervention of an attacker. Once an anomaly is detected, an investigation must be conducted to determine the source of the anomalous behaviour.

A digital twin (DT) is a virtual representation of a physical system [33, 110, 197]. It is a highly advanced software tool created through the careful collation, analysis, and understanding of the data produced by the asset being twinned [83]. DTs enable advanced inspection of their physical asset's design and behaviours and predict how it might behave under future conditions [89].

DTs can model and transform system data into information about the state of the system being twinned, helping operators understand their systems better. Depending on the needs of the system operators, this information can take many forms. One area of interest in DT development is their potential application towards cyber security, assisting system users in detecting attacks and preventing attacks [98]. Since a DT is developed to understand the design and behaviours of the underlying system, it is theoretically well-placed to identify when that system is not behaving as intended.

The challenge with the development and integration of DTs is that they are a new technology. It is not yet well understood how DTs should be developed and integrated into such a large and critically important system as an ICS. If the models within the DT are prone to bugs or can be manipulated, this can create misinformation for operators, leading them to mismanage the system. Similarly, if the system connecting the DT to the twinned system can be manipulated to obscure the actual state of the twinned systems, this can also lead the DT to produce misinformation. Although DTs are an emerging technology, they are not yet widely integrated into critical system operations. However, as their benefits become more evident, their deployment in increasingly sensitive systems will grow, ultimately leading to their adoption in critical infrastructure. Ensuring the reliable and secure development of these systems is a crucial challenge that must be addressed alongside advancements in their utility.

One approach to address these concerns is through the application of formal methods. Formal methods use mathematically rigorous techniques to analyse a system and can be used to provide assurances about the soundness of its design. These techniques have been applied in related domains such as Cyber-Physical Systems (CPSs) and the Internet of Things (IoT) to identify unsafe conditions and reason about system correctness with respect to security requirements. When applied to the technologies used in the construction of a digital twin, formal methods can be applied to verify that collected system data corresponds to expected behaviours and support the integration of DTs into ICS environments by identifying cybersecurity vulnerabilities.

Despite the increasing interest in and development of DTs in industry, the technologies in-

volved in their creation are at varying stages of maturity. Each component may be implemented in many different ways depending on the requirements of the DT. Finding suitable combinations of technologies that work together in a cohesive system is a key factor in the maturation of this type of system. Additionally, a wide-reaching sociotechnical digitalisation process is required to translate the user knowledge and processes of a real-world physical entity into a reliable, data-driven structure that DTs can integrate with. This is a highly complicated task requiring organisational restructuring, retraining of workers, and thorough collaboration across work domains throughout the process. For many organisations, this will be a slow evolution that is likely already underway, with the functionality of DTs being developed and integrated into the workflow over time. It is therefore essential to understand how these systems should be designed to manage expectations and risk during this transformation period and beyond.

We propose that by focusing on the operational requirements of DTS and the threats that may undermine them, we can develop an appropriate approach to inform the secure design of DTs. We further propose that formal methods can be used to assess the design of DTs with respect to these operational requirements and can support overall system resilience by performing intrusion detection.

1.2 Contributions

Our key contributions are as follows:

1. A case study developing the infrastructure to integrate a DT with a testbed representation of a hydroelectric dam (Chapter 4).

2. The development of a formal methods-based anomaly detection method (Chapter 5). The approach utilises Promela representations of PLC ladder logic and the SPIN model checker to verify the validity of system states with respect to the previously observed system state.

3. A series of methods to represent DT infrastructure in the Alloy specification language (Chapter 6). This culminates in the development of a novel approach to specifying an action-based attacker as they infiltrate a system's security. The developed system contextualises the impact of this with respect to the DT and its ability to replicate the underlying twinned system accurately.

1.2.1 Thesis Statement

Thesis title: *Applications of Model Checking in the Context of Cyber Security for Digital Twins*

Thesis statement: Model checking can support the secure development and integration of DTs within ICSs. We demonstrate this through the integration of the SPIN model checker in a hydroelectric dam DT to detect behaviour anomalies and the subsequent security evaluation of the integration using the Alloy model checker.

1.2.2 Funding

I was funded by EPSRC Industrial Case account EP/V519686/1 and was sponsored and co-supervised by Andy Deacon and Colin Thomas at the Defence Science and Technology Laboratory (Dstl).

1.2.3 Associated Publications

Some of the material in this thesis is to be published in future papers:

- The background material and example introducing SPIN in Chapter 3 is included in [164].
- The framework developed in Chapter 4 and the integration of the SPIN model checker have been submitted to Formal Methods for Industrial Critical Systems (FMICS) 2025.
- The network modelling approach of Chapter 6 is in preparation for submission to Formal Methods in System Design, 2025.

1.2.4 Repository

For the interested reader, an archive containing all models and code used in this thesis is available online at: `doi.org/10.5281/zenodo.15482551`. Further details on its content are provided in Appendix A.

1.3 Structure of Thesis

The structure of this thesis is as follows:

- Chapter 2 gives an overview of ICS and the cybersecurity threats and challenges they face. It also provides some background on DTs and an introduction to formal methods and their application to DTs.
- A detailed introduction to formal methods and our tools, in particular the two model checkers that we use (Spin and Alloy), is presented in Chapter 3.

- In Chapter 4, we detail the construction of a DT of a hydroelectric dam for collecting and monitoring system data. We develop a historian and measures to synchronise data from the monitored devices to create a unified system state snapshot.
- Our approach, using the Promela specification language to model behaviours of the hydroelectric dam and the Spin model checker for anomaly detection, is presented in Chapter 5.
- In Chapter 6, we design an approach using Alloy to evaluate the security of our constructed hydroelectric DT framework.
- In Chapter 7, we present our conclusions and discuss future work.

Chapter 2

Background

In this chapter, we present an overview of the literature surrounding DTs and their use in the domains of ICS and cyber security. We consider the domain of ICS, where DTs have been proposed as a solution for various security challenges. We give an overview of what ICSs are and the technologies used within them. We then consider the challenges facing these systems and discuss the ways that DTs can help address them. An explanation of DTs is presented, detailing their origin along with a description of their key characteristics, components and behaviours. We then consider how DTs can help with the security of ICS by reviewing some of the current guidance for securing these environments.

2.1 Industrial Control Systems

Many services and critical systems in modern society are operated by or depend upon ICS. Industries such as energy production, water treatment, manufacturing and maritime have evolved a specific set of requirements that require specialised computer hardware. In these environments, the systems must be able to respond rapidly to changes in environmental conditions to maintain a stable system through feedback control. The technologies used to perform this function at scale have a directly line of evolutionary innovation tracing back to The Industrial Revolution.

2.1.1 Evolution of Modern ICS

Modern ICSs have developed as a result of developments to increase productivity, efficiency, safety and to enable better operational management. Traditionally, historians classify the Industrial Revolution as a single period of industrial innovation. However, analysis from other disciplines in economics and technology have redefined this into three distinct industrial revolutions, classified by the innovation trends within each [193]. Within this framework, it is argued that we are now at the outset of a fourth industrial revolution, often termed Industry 4.0 [193, 194], characterised by the integration of Cyber-Physical Systems (CPS), the Internet

of Things (IoT), big data and AI-driven automation [78, 120, 185]. DTs are expected to play a crucial role in Industry 4.0, harnessing increased data collection to connect the physical world with AI [78]. To better understand the future of modern ICSs, it is helpful to examine some of the key innovations that shape the way they are used today.

A feedback control system responds to changes in its environment to maintain a desirable condition in equilibrium. It does this by executing a stabilizing automatic response that is functionally proportioned to the size of the environmental deviation from the desired state. As animals, many of our natural behaviours can be considered as feedback control mechanisms that ensure our continued survival: finding warmth when we are cold, water when we are thirsty and food when we are hungry. As human society has evolved, so have our methods to automate these mechanisms, enabling us to spend more time on other pursuits.

Historians have found evidence from over 2000 years ago of Greek scholars developing precursor components for feedback control, such as water clocks for time measurement and wine dispensers for controlled liquid flow [26]. Their mechanisms would then inspire refinements in Arabic books from the ninth and thirteenth centuries, demonstrating more recognisable feedback control applications for water dispensing. However, it was not until the late 18th century that the float valve level regulator was reinvented, and these concepts saw widespread industrial-scale adoption. Around the same time, early industrial thermostats emerged, and millwrights in Scotland and England developed windmill speed regulators to reduce millstone wear [158]. These experiments led to the development of the lift-tenter device, which later inspired Scottish engineer James Watt to use centrifugal governors in steam engines that would be used as power sources throughout the rest of the Industrial Revolution.

During the early stages of the Industrial Revolution of the late 18th century, the development of steam and water power enabled many processes that had once been conducted by hand to now be performed by machine. Electrification then enabled the development of the modern production line where machines could be placed in sequence for the needs of the goods being manufactured without focusing on the location of steam power sources. The Bessemer process increased steel production, expanding railway networks with more durable tracks. These railways facilitated the movement of people, resources, and ideas, accelerating industrial growth. It is from the increased organisational needs of these railway networks that modern business management hierarchies and groupings of individuals into clear departments emerged. Finally, the development of the electric telegraph enabled long-distance communication of text messages, seeing rapid adoption initially along railways for signalling and keeping track of carriages.

The discovery of telephony brought with it the development of analogue electrical circuitry that could be used to process feedback signals from early sensors. The invention of the negative feedback controller enabled feedback control to solve many non-linear problems, leading to its widespread use during the First and Second World Wars. This period saw the development of feedback control into the broader field of control theory, which was applied extensively in war

technologies such as anti-aircraft guns, torpedoes and control relays [26]. Increased knowledge-sharing after the war led to the development of modern control theory algorithms, including the Kalman [238] and Bellman [118] filters that can approximate unknown system parameters from measured data.

The development of computers allowed for calculating previously unknown system parameters through simulation modelling. Computers were initially integrated into process control as supervisory systems that optimised processes controlled by more traditional relay systems. However, with the introduction of the microprocessor in the 1970s, the computer became the primary process controller in many domains. The introduction of microprocessors began the generalisation of the ICS. In modern systems, while different sensors and actuators that are specific to the needs of the process are used, the devices that gather and provide control instructions remain consistent. This enables the creation of standardised supervision and management systems that support the operation of these microprocessors.

Since the development of the generalisable process controller, ICS processes have become larger and more complex. Our ability to understand and simulate the process continues to grow, providing safety and efficiency benefits. Many different vendors support the development of ICS technologies; while these initially operated on proprietary standards and protocols, progress is being made towards interoperability. This increased interoperability and communication between devices across a system is a key enabler of Industry 4.0 technologies. These technologies seek to leverage this increased data sharing towards a better understanding of not only the physical processes they control but also the broader sociotechnical challenges associated with their management and operation.

The biggest challenges faced by a modern ICS are no longer deficiencies in the physical control systems but rather organisational complexities, managing assets throughout their lifecycle, incorporating sustainability, and collaboration across stakeholders. The lifecycle of ICS components now extends beyond traditional operational concerns, requiring careful consideration of upgradability and cyber resilience during system design. Additionally, the move towards greener practices creates pressures for increased energy efficiency, waste reduction, and circular economy practices. Finally, as industries move towards fully integrated supply chains and cloud-connected business operations, effective collaboration between organisations is essential to the continued operation of the ICS. As we will discuss in Section 2.3, these are all challenges that DTs play a role in addressing.

2.1.2 Structure

The evolution from relay-based controllers to modern microprocessor-based controllers transformed the capabilities of industrial control, but it also facilitated the development of a consistent architectural structure of components.

The instrumentation and field devices used in each domain may differ; however, the com-

ponents controlling them are broadly similar. The structure of ICS networks is hierarchical, separating the Operational Technology (OT) systems that directly control and measure the industrial process from the Information Technology (IT) systems used by employees to perform other tasks in the organisation. Previously, this would have been achieved by keeping the OT devices entirely offline, such that there was no interconnection between IT enterprise networks and OT devices. However, as the needs of modern ICSs evolve, a means of sharing data more efficiently between the two types of network is required. To do this securely, the network must be segmented to avoid deploying vulnerable OT devices alongside internet-connected IT systems. This type of network segmentation is most often achieved through applying an adapted version of the Purdue Enterprise Reference Architecture [239] which was originally developed in 1994, since dubbed the Purdue Model. The Purdue Model, shown in Fig. 2.1, consists of 5 levels of devices, beginning with the OT components at the lowest level and building up to the wider enterprise network at levels 4/5. At each level, a set of devices is grouped together according to their type and function.

Level 0: Field Devices

This level contains the sensors and actuators that gather data from the process and directly manipulate it. These field devices are the front-line, process-facing elements of the system. Examples of sensors used in an ICS are volume sensors, pressure gauges, temperature sensors, wind-speed and directional sensors, proximity sensors, and video feeds. These components are often directly wired into a local controller, whereby they communicate sensor data or receive instructions. Traditionally, this is done by digital signalling or through analogue variance of the voltage, current or resistance on the connection to the local controllers. However, modern ICS systems increasingly use a variety of standardised and proprietary communication protocols to exchange data with field devices.

Level 1: Local Controllers

Field devices are directly connected to local control devices located in level 1. Programmable Logic Controllers (PLCs) are industrial computers designed for monitoring and controlling industrial processes. They are general purpose, being easily reprogrammable allows them to be applied in a wide variety of environments where computational outputs must be calculated within a strict time window after receiving inputs. The PLC addresses this explicitly through its design. Each PLC executes a "scan cycle" whereby during each execution, it reads all inputs, performs a predefined set of operations and then writes a set of outputs that can then be affected by actuators. The time taken for an entire cycle of this varies depending upon the length of the code written to the PLC; however, it typically ranges between 10-150 milliseconds.

OT communication networks facilitate high-speed, real-time communication to provide controllers with the most up-to-date data about the controlled process. However, many of the pro-

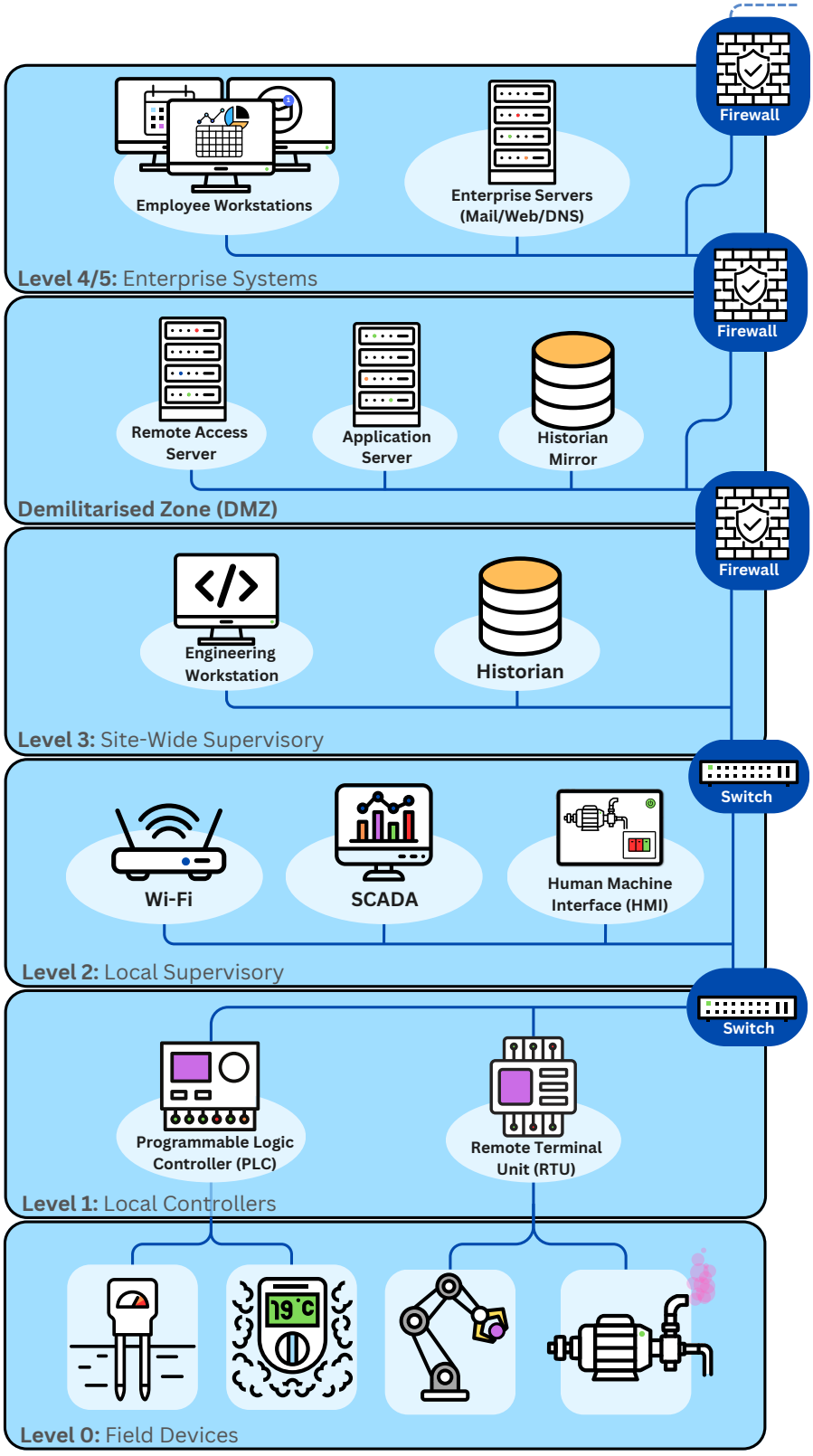


Figure 2.1: The Purdue Model, structuring the network architecture of an ICS to separate vulnerable OT systems from IT networks while still providing connectivity through a firewalled DMZ.

tocols still in use today were designed when ICS environments were entirely disconnected from wider networks. As a result, there was no need to incorporate security measures into their design. As we explain in this section, ICS networks are no longer separate from their wider corporate infrastructure, and as we'll discuss in Section 2.2.1, these vulnerable communication protocols make it easy for attackers with access to an ICS network to undermine the system. Maintaining these devices in their own separate network area allows for the specific proprietary protocols that OT devices use without inadvertent interference from IT protocols. It also allows for easier identification of malicious network traffic outside the local control level.

Level 2: Local Supervisory

Local controllers are connected to local supervision systems in level 2 through a switch. Supervisory Control and Data Acquisition (SCADA) systems and Human Machine Interfaces (HMI) are used for ICS environment supervision. These devices are used for vital data acquisition tasks that enable operators to monitor the operation of the plant through SCADA [17]. At a local supervisory level, these devices gather, collate and store data from the controllers within a logically defined subsection of the ICS, allowing system operators to monitor that area of the process. If the system's behaviour needs to be altered, HMIs can be used to provide manual or automatic control inputs to the local controllers.

The supervisory systems need to communicate with the process control devices to gather data from them, while still keeping the majority of their traffic separate. Virtual Local Area Networks (VLANs) are often used to do this by segmenting the network. This allows for traffic from devices that share the same switch to be logically separated into distinct groups of devices, enforcing the segmentation of the Purdue model. This allows supervisory data to be shared to displays and forwarded onto devices at higher levels without introducing additional traffic at the process control level.

Level 3: Site-Wide Supervisory

SCADA systems often utilise a hierarchical structure. Local supervisory devices are, therefore, often connected upwards to site-wide supervisory devices at level 3. Historians are databases used to collect and store process data and may be used to gather data from different sections of the system. While many of the technologies used at level three may be the same, the difference between levels 2 and 3 is scope. At level 2 the devices focused on the controllers of a specific aspect of the process; at level 3, the supervisory data from across the ICS environment is collected to provide oversight into all aspects of the site's operation. If modifications to the operation of the local controllers are needed, these are made through engineering workstations, which may be located at either level 2 or 3, depending upon the size of the site. Again this segmentation has dual benefits; for the systems operating at level 3 it reduces traffic at their site-wide level, and

for the devices operating at level 2 it prevents systems in other plant areas from interfering with their operation.

Demilitarised Zone

One of the primary features of the Purdue Model is the implementation of a Demilitarised Zone (DMZ) that divides the network. The DMZ allows users and devices operating in the enterprise IT network to gather data from the devices in the OT control network. Traffic from the enterprise networks at levels 4 and 5 cannot pass through the firewall from the DMZ into level 3. Conversely, traffic from the control network cannot level the DMZ to reach the IT enterprise network. Configuring the firewall in this way effectively constructs an artificial "air gap" between the two networks while still allowing indirect communication through the overlapping intermediary services located within the DMZ. This is important since the enterprise network is connected to the internet and, therefore, is at much greater risk of compromise. Historians located at lower levels share data with DMZ-hosted mirrored copies, allowing enterprise users to query historian data without ever interacting with the historian directly. Applications can be hosted on servers within the DMZ, allowing for services useful to both the OT and the IT network to be shared. If authorised users in the enterprise system require access to a device in the control network, they can access it through remote access servers hosted in the DMZ since their actions will originate from within the DMZ, not the enterprise network.

Level 4/5: Enterprise Systems

To complete the overview of the Purdue Model, level 4 consists of site-specific business devices and services used by the users on site. These may be administrator devices used to manage the people and systems on-site, or they may be users performing analytics on performance who do not need access to the low-level devices to carry out their roles. At level 5 are the organisation-wide services, such as corporate servers hosting web and mail services that may be accessed by users in sites across the organisation or from outside the sites. While these systems usually should not directly interact with the ICS. Their ability to impact the operational systems controlling the ICS fundamentally affects how a DT can be deployed within an ICS environment. As we will discuss in detail in Section 2.3, a DT requires real-time access to data from control systems. However, as a key digitising technology, many of its users will be located within the enterprise network. A major challenge in deploying a DT to an ICS is securely extracting data from vulnerable OT devices, processing it, and providing enterprise users with access without compromising OT network security.

Deploying the Purdue model provides in-depth defence for ICS operators, hiding the most vulnerable OT devices at the furthest point from outside attackers. It allows authorised users within the organisation to access the services, data and devices that they need while imposing logical separations that help to prevent accidental or deliberate interference from within the

network. In addition to these security benefits, network segmentation provides performance benefits, reducing traffic at each level and helping to maintain a measure of separation between IT and OT communication protocols. As OT networks are currently implemented, these benefits make using the Purdue model indispensable as a tool for structuring ICS networks. However, even with these defence in depth measures in place there are many examples of ICS environments being attacked, as we will discuss in the next section.

2.2 ICS Cyber Security

ICSs have evolved to meet specific operational requirements derived from the needs of the industrial processes that they control and the trusted operators working to manage that process. They are used in various sectors of modern infrastructure, including energy, manufacturing, defence, transportation, and water and wastewater treatment. Since processes monitored by ICSs can change rapidly, the system requires high availability to respond quickly. If conditions are not monitored carefully, the operational environment in which these systems work can become highly dangerous, causing damage to the system and endangering the operators' lives. This also makes these systems resistant to change, often preferring tried-and-tested legacy systems, networking devices and protocols over integrating new technologies that may cause unpredictable consequences. Additionally, the need to prioritise system and user safety means that some security processes that are commonplace in an IT or enterprise systems environment are not utilised within these OT environments. In some circumstances, a difficult judgment on the trade-off between safety and security must made.

Cyberattacks are an increasingly common occurrence for organisations across the globe. At the beginning of 2024, 50% of UK businesses and 32% of charities reported some form of cyber breach in the last 12 months, with notably higher rates for large businesses [67]. The ability for the perpetrators of cyber attacks to remain anonymous while causing considerable damage to their targets at a relatively low cost makes them an attractive option for adversaries wishing to interfere or spy on the operations of modern organisations. Cyber attacks can be carried out by a wide range of groups with varying resources. Nation-states [206], cyber terrorists and other malicious parties all engage in covert operations. Industrial and national dependence on ICSs makes them intrinsically valuable as targets for attacks from these groups, as can be illustrated by a recent history of increasing attacks against them.

2.2.1 Cyber Attacks

A cyber security threat occurs when a threat agent is able to detect a vulnerability (a flaw or weakness in design) and exploit it to further their objectives. The connection of ICS networks to enterprise networks used for business management, while beneficial for system management, proves to be a significant vulnerability for the cyber security of ICSs that dramatically increases

the attack surface [191]. Though typically separated by a firewall or DMZ, many examples of attacks permeating through these protections can be seen. Initially, successful attacks of this nature appeared to be accidental, collateral damage of a non-specific piece of malware infecting a device on the enterprise network and spreading to the ICS system. In recent years, however, these have progressed to deliberate attempts to gain access to the ICS system through the enterprise network. In this way, the enterprise network can function as a stepping stone through which logins can be acquired to access the control network [202]. A timeline of major reported cyber attacks is presented in Fig. 2.2 alongside prominent cyber defence guidance released in response.

Early Incidents: Accidental Infections

In the early 2000s ICS attacks were often collateral damage of generic malware infections. In February 2003, at the David-Besse nuclear power station a contractor's computer connected to the internet outside of the firewall, allowing the Sapphire computer worm to compromise the network. The worm targeted unpatched Windows hosts and couldn't target OT devices, but it caused significant disruption by infecting the safety parameter display system and the process control network. Similarly, in 2005 when an internet worm infected the Windows hosts at the Daimer-Chrysler car manufacturing plant causing the plant to shut down for an hour. The incidental nature of these attacks demonstrates that even internet-based malware without a specific target can access and disrupt air-gapped ICS networks.

Stuxnet: Introduction of ICS Malware

In 2010 the threat to ICS systems increased after Stuxnet became the first intentional cyber attack against an ICS, resulting in physical damage to the site [11]. Malware was used to damage the Nantaz uranium enrichment plant in Iran by infecting Microsoft Windows operating systems used to operate SCADA systems. The attack is believed to have been carried out by both the United States and Israel in a calculated effort to undermine Iran's nuclear development. It is notable for the fact that it spread indiscriminately through infecting removable drives and across private networks while remaining inert unless the very specific configuration requirements for the attack were detected [145].

Increase in ICS Attacks

Stuxnet showed the world the extent of the damage that targeted attacks on ICS networks could cause. It is notable for being the first publicly known, purpose-built ICS Malware designed to target process controllers by compromising SCADA systems. Since then, a dramatic increase in reports of ICS cyber attacks has been observed [30], with significant attacks being recorded in each year between 2014 and 2021 [11]. Among these attacks, only four other pieces of ICS-

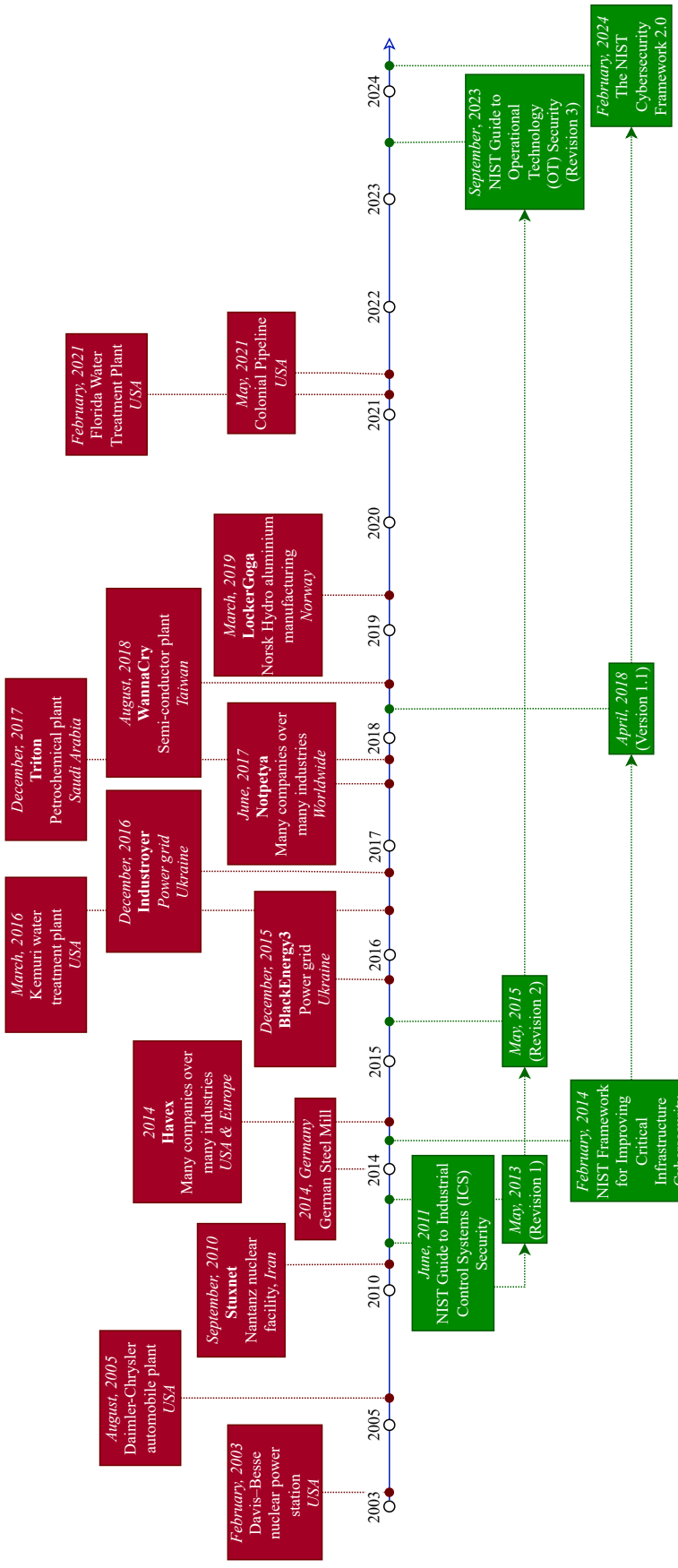


Figure 2.2: Timeline of attacks on ICS infrastructure alongside notable developments in cyber defence.

specific malware have been publicly documented; others may exist already and yet more are likely to be developed in the future.

In 2014, an unidentified German steel mill was reported to have suffered a spearfishing attack that compromised the mill's business network before crossing over to the control network and causing a blast furnace to shut down incorrectly, causing physical damage. In the same year, the Havex malware was used extensively in spearfishing attacks by a Russian APT group to target ICS networks across aviation, defence, pharmaceutical, and energy sectors, affecting an estimated 2,000 sites across North America and Europe. Havex was notable for its ability to map and identify specific ICS devices by scanning TCP ports commonly associated with industrial traffic. It demonstrated the devastating cross-domain potential of developing malware for ICS networks as similar components are used in systems across industries.

Cyber attacks are an increasingly integral part of modern conflict; nation-states worldwide engage in cyber warfare to disrupt critical infrastructure, steal intelligence, and undermine adversaries while avoiding conventional military engagement. Russia's repeated deployment of cyber attacks against Ukraine as part of the ongoing Russo-Ukrainian War is an example of this. In 2015, Russian hackers used BlackEnergy [206], the third publicly known ICS-specific malware, to disrupt Ukrainian power stations, cutting electricity to 225,000 customers for 6 hours [124]. A year later, the same group deployed Industroyer (CRASHOVERRIDE) [202], the first malware designed specifically for power grids, leveraging native ICS communication protocols to send malicious instructions [149]. A new variant appeared in 2022, dubbed Industroyer2, causing more blackouts in Ukraine [218].

The fifth and final publicly-known ICS malware is Triton [58]. Discovered in a Saudi Arabian petrochemical plant in 2017 it specifically targetted the safety systems, instead of the process controllers. The malware attempted to reprogram the safety controllers to disable safety failsafes, potentially leading to dangerous, uncontrolled industrial conditions. Russia has also been implicated in this attack.

2.2.2 Threat Intelligence

To help inform the development of defence and response activities, attempts have been made to understand and track the activities undertaken by attackers.

The Cyber Kill Chain model [104] gives a 7-step description of the behaviours of an attacker, particularly an APT, when attacking a network. An attack begins with reconnaissance, where the threat actor gathers accessible data that may be used to gain access to the system and uncover details about the internal configurations that may be useful once access has been achieved. This might involve accessing leaked or previously acquired user data and identifying the structure of the internal network and the components within an ICS plant from publicly available documents. Once the attacker knows the type of system they are attacking and has determined the method through which they will gain access, they begin the process of weaponisation. The exploits that

allow the attacker to target a component within the identified system are integrated into a means of delivery to create a package capable of gaining entry to the system, once delivered. Delivery of the weaponised package is then used to send the weaponised exploit to the target. Once at its intended destination, it exploits a vulnerability to install malware on the target system. From this point, the attacker begins the process of command and control to cement its hold on the system and mask its presence while establishing remote access to the system for themselves. At this point, the attacker has accessed the system and is free to use that access to achieve objectives to the best of their ability. This may lead the attacker to remain dormant, observing, gathering reconnaissance for other attacks or potentially waiting for a trigger to attack the system.

Analysis using the Cyber Kill Chain model can be further enhanced using The MITRE ATT&CK framework [48]. This is a continuously updated knowledge base derived from the Cyber Kill Chain model. It consists of categorisation and documentation of the attack techniques used at each stage of known attacks on an enterprise, mobile, or ICS system. Each technique is linked to specific examples of malware and APT groups that have used the specific technique. By combining these two, it is possible to construct a lens through which to view a system when attempting to analyse it for vulnerabilities. Remaining aware of new threats and applying the knowledge gained from previously executed attacks is key to developing better cyber security practices and enhancing the security of critical systems.

2.2.3 Cyber Defence

The increasing frequency and potential impact of these attacks have prompted responses at multiple levels. Cyber security has developed rapidly as a field of research backed by considerable investment. Political investment from nation-states in cyber security expenditure and personnel has accelerated in a manner characterizing a “cyber arms race” [51, 116]. The perceived offensive advantage of cyber warfare intensifies the escalation similar to that of the nuclear arms race of the 20th century, where mutually assured destruction was similarly perceived as the most effective method of deterrence. However, cyber warfare’s secretive, amorphous nature makes it considerably more challenging to develop arms control for these intangible cyber weapons [73, 79].

Given the persistent threat of cyberattacks and the rise of cyber warfare, ICS stakeholders must take proactive measures to defend their systems. While IT security measures are well-established, OT security is still evolving. Achieving total security for internet-connected ICS devices is often unrealistic, especially against well-resourced adversaries, including nation-states. Additionally, security measures must balance protection with operational safety, as some critical infrastructure may need to prioritise availability even during an attack. This has driven the development of intrusion-tolerant response strategies [102]. As a result, operators face the challenge of adapting ICS environments to counter modern cyber threats they were never originally designed to withstand.

There have been several key pieces of published guidance to support the cyber security response to this increased threat. On February 12th 2013, President Barack Obama signed Executive Order 13636, citing the need for improved cyber security following "repeated cyber intrusions into critical infrastructure". The order outlines a set of measures to deliver a cyber security framework to enable owners and operators of critical infrastructure to identify, assess and manage cyber risk [214]. A year later, the first version of the NIST Cybersecurity Framework (CSF) was published [174], presenting a categorisation of the measures organisations should take to manage cyber security risks.

ISA/IEC 62443 [173] is a series of international standards specifying requirements for key stakeholders at all levels of ICS deployment and operation. The series requires the establishment of a control systems security program [106] that assesses risks based on the criticality of system components. It also sets technical requirements of the system [107] components [108].

NIST Special Publication 800

The NIST Special Publication 800 series [208] is a guide for how to secure OT systems. It advises how to establish a cyber security program within an organisation, how to use that program to assess and manage risks to ICS systems and how to integrate security into the design of an OT network, particularly through the application of network segmentation techniques such as those described in the Purdue model. The most recent version is the NIST Cyber Security Framework (CSF) 2.0 [175]. We will use this framework to contextualise the different cyber defence activities that ICS shareholders use to secure their systems. In doing so we will address the other guidance documents where relevant, and reference relevant cyber defence publications.

The first step in ICS security is understanding organizational risks by identifying assets, assessing vulnerabilities, and determining areas for improvement. Threat intelligence tools like MITRE ATTCK help minimise adversaries' advantages by increasing awareness of modern attack techniques. Analysts should ensure components follow security standards like ISA/IEC 62443.

The NIST SP 800 series outlines a four-step risk assessment process—framing, assessing, responding, and monitoring—aligning with the CSF identification phase, which considers regulatory requirements and risk tolerance. While eliminating all security risks may be unrealistic due to cost and safety trade-offs, ICS security must balance accessibility for safety mechanisms with protection from cyber threats. Ultimately, organizations must decide whether to accept, transfer, share, or mitigate risks based on their assessments.

Once risks have been identified appropriate measures are taken to defend the identified vulnerable assets from attack. These encompass a variety of activities across the socio-technical spectrum of an organisation. This includes improving device resilience, upskilling staff with cyber awareness training and continuing to develop the organisation's cyber security program. The NIST Special Publication 800 series details some best practices for implementing technical

protection measures, building upon each as a layered system. The first two layers are developing the cyber security program and physically securing the site and its devices. Cyber security practices to improve the devices' resilience begin with establishing a network segmentation architecture. Further activities increase network security by instigating monitoring and logging and the development of a zero-trust architecture. A zero trust architecture uses authentication and authorisation techniques throughout a system, regardless of how hard-to-reach a network segment may be or how trusted the device that the user is using is [205]. The final layers focus on improving hardware security – through tools such as access control, monitoring, and device identification – and software security by ensuring software is kept up to date with regular patching and application hardening.

With protection measures in place, the next step of the defence process is to instigate detection activities that identify attacks. These comprise continuous monitoring for anomalous cyber security events to trigger response actions from operators if an attack is detected. Detection extends to many facets of the built environment, but it is enabled by monitoring the systems instilled within the ICS in terms of network activity and device behaviours. Monitoring approaches vary in the intensity with which they are integrated into the monitored system to acquire network traffic data, either using passive network taps or directly integrating into the network to communicate directly with the systems they monitor. These monitoring systems often need to have a light footprint on the network that they monitor due to how easily disturbed legacy ICS networks can be; if a particularly intensive monitoring system is added to one of these networks, it can disrupt less robust local controllers.

Intrusion Detection Systems (IDSs) utilise various techniques to identify intrusions in monitored systems. Signature-based detection methods attempt to identify known attack patterns in network traffic. Anomaly detection techniques establish a network activity baseline and then flag behaviours and exchanges that are outliners from this baseline. Finally, host-based methods focus on a device within the environment to identify unusual behaviour patterns for that specific host.

Network-based IDS approaches are effective in ICS environments due to the more consistent nature of OT network traffic. Patterns in communication can more easily be established, allowing IDS systems to determine an operational baseline from which deviations can be identified [244]. IDSs of this type have become a significant area of interest for machine learning (ML) algorithms, particularly supervised learning approaches [222]. ML approaches are well suited to identifying deviations in datasets, particularly when provided with labels that identify normal behaviour and abnormal behaviour. Using previously observed system behaviour to establish a baseline works well in simple environments; however, there are challenges associated with the use of ML. The vastly higher quantities of normal behaviour compared to adversarial behaviour result in a skewed balance of accuracy when labelling observations, making it harder to determine the true efficacy of a ML IDS algorithm. It has also been demonstrated that when

an attacker knows the system being attacked, they may be able to design adversarial ML algorithms that can evade detection of ML algorithms [249]. Therefore, while ML algorithms are well suited to network layer intrusions, they are less suited to detecting behaviour deviations in a physical process.

An alternative approach that is better suited to deviations at a process level is specification-based intrusion detection [127]. This uses a formal specification of the underlying system to define normal and abnormal behaviour [196]. In [4], an invariant-based approach is used to derive the value of sensor data from alternative sensor values, enabling the detection of attacks. However, due to the approach integrating the detection equations into the PLC, if the attacker can reset the PLC, it can interrupt the detection, causing attacks to evade detection. It has been suggested that IDSs could run alongside or be integrated into DTs, leveraging their knowledge of the system's design to identify abnormal behaviours [191].

Regardless of the intrusion detection method employed, once an incident has been identified, the remaining two themes, respond and recover, trigger response activities to identify and combat the threat while taking actions to regain control of the system. The first step is to identify whether the detected incident is a legitimate attack or a false positive caused by legitimate user behaviour or a bug in the system. In a large system, there can be large quantities of these false-positive cyber incidents, making identifying legitimate attacks challenging [37]. If a legitimate attack has been identified, response procedures compromise establishing the extent of the attack and containing it. Once an incident has been contained, the recovery stage promotes the eradication of the compromise and the initiation of forensic processes.

2.3 Digital Twins

Taken at its most basic concept, a DT is a digital representation of a real-world entity or system [110]. The value proposition of a DT is primarily that simulation and inspection tasks can be performed far more quickly and efficiently on a DT than they can on the physical asset that the DT represents. Additionally, while a physical asset can have high construction and maintenance costs and may be damaged during testing, a DT of that same asset incurs minimal maintenance costs, can be infinitely instantiated and can be simulated in hazardous scenarios without any risks of damage or loss. Therefore, while the construction of a DT currently presents challenges in the short term, the long-term benefits of the concepts have attracted considerable attention from parties across many industries. In an increasingly data-driven era, organisations across diverse sectors are eager to leverage large volumes of data towards better decision-making and operational management of assets and processes. In a digital environment, data can be more readily processed and disseminated within the organisation without requiring access to the physical system. As a result, there is a tendency to view DTs as a magic bullet, providing a readily examinable medium for human operators across departments to collaborate while also being in

a digital format that can be processed by ML algorithms and integrated with AI tools. DTs are widely considered a key enabling tool to streamline decision-making in the operation and management of physical systems, particularly with the potential to integrate them with increasingly powerful AI workflows. However, perhaps due to this widespread interest across sectors, DTs have become an industrial buzzword, confusing what exactly a DT is and how they should be designed.

In this section, we define what a DT is by reviewing definitions found in the literature and considering the evolution of the concept. In doing so, we will explain some related ideas often used alongside or within DTs and delineate how they differ. We review how DTs are used in the domain of ICS and explain how their unique characteristics lead to interesting considerations for how to integrate DTs with ICS. This leads us to review some of the current discussions about the implications that connecting a DT to an ICS environment could have to either bolster or undermine the cyber security of the original system.

2.3.1 Origin

Although the term DT is relatively new and can be readily attributed to an influential 2010 NASA Roadmap [182, 198], the idea's origin is less clearly understood. DT is a concept that has emerged from a desire for innovative Product Lifecycle Management (PLM) approaches. It's also a concept that became conceivable through the maturation and combination of several enabling technologies in simulation, data management, communications and IoT. Sources do not agree on whether the concept was published first and then began inspiring the application of the technologies towards its implementation or if these technologies were already being used towards this objective without a specific name being attributed to it. As a result, within the literature two commonly cited DT origins can be found.

The first formal publication of the concept, albeit without the use of the term DT, has been attributed to a 2002 University of Michigan presentation by Michael Grieves [88] (shown in Fig. 2.3). Of particular note is the proposition of using multiple virtual spaces, built using data from a "real space", for better PLM. However, some have argued that the DT paradigm had been practically applied decades before during the Apollo missions of the early 1970s [190], most notably Apollo 13. Fifteen high-fidelity simulations of the spacecraft were interlinked for use during the planning and execution of the mission, ultimately playing a key role when critical system failures to the spacecraft required urgent changes to the trajectory of the flight during the mission [70]. Grieves has argued that the simulations were closer to a network of physical models and that the use of "primitive computing components" is merely an evolution of the long-applied technique of physical twinning [89]. This disconnect between conceptual ideal and pragmatic implementation of DTs is common in DT discourse and can lead to both scepticism and unrealistic expectations about the potential of DTs.

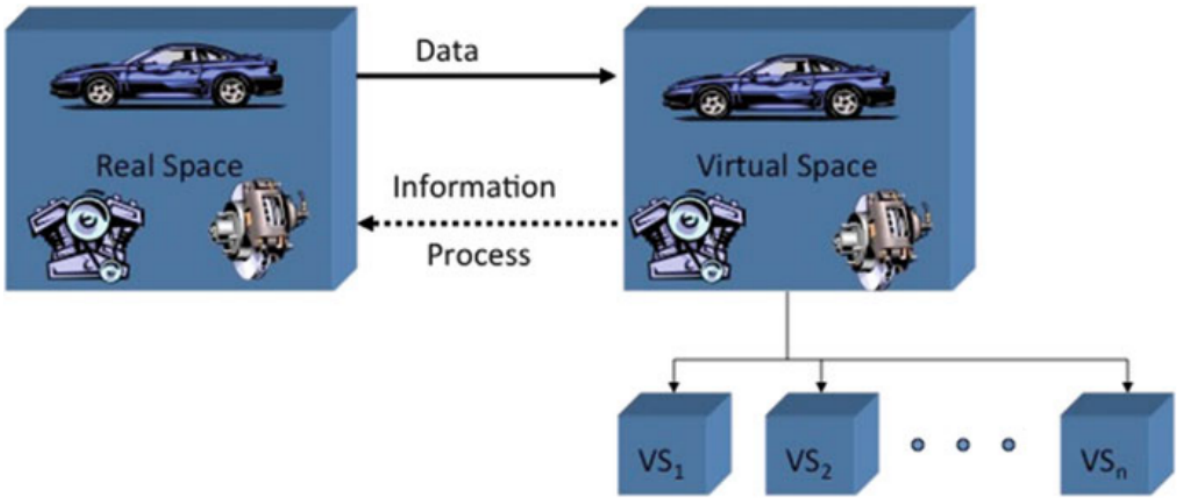


Figure 2.3: "Conceptual model for PLM" slide showing virtualisation of a physical asset, from [88]

2.3.2 Definitions

Many definitions of DT are used throughout industry and academia. Literature reviews of DTs and their definitions demonstrate that the term “digital twin” has been continually refined as the concept continues to evolve within several application areas [150,197]. As the technologies used in implementing a DT mature, so does the understanding of what can be achieved when they are combined. While the physical subject of the DT varies across domains, having a notable effect on implementation details, a common definition of what a DT is has been settled upon. Notably, in the emergence of this definition a number of other concepts have also been identified, in some cases as a means of defining the DT by what it is not. Prominent definitions for DT and related concepts that are differentiated by the characteristics of the system are presented in Table 2.1

The ongoing redefining of the term and the creation of similar yet distinct definitions reflect a desire to more clearly characterise an ideological concept in terms that can be practically understood and implemented. Each of the terms presented in Table 2.1 reflects a subtly different interpretation of what it means to replicate a physical object. Some argue that the digital model, digital shadow, and DT itself can all be classified as subcategories of DT [130], implying that there exists some overarching concept of the DT system that varies by purpose. It has been argued that this categorisation is overly reductive, as these terms are primarily differentiated by their connectivity to the physical system rather than the accuracy of their representation [87].

However, these terms describe ideas that certainly combine within the DT paradigm, a DT encapsulates a digital model and a digital thread. A major challenge in creating a DT-type system is in identifying which of these ideas is most appropriate for the given use case. The boundary between digital shadow and DT is often cited as a difference in how the data is transferred from the digital representation back to the physical twin. A digital shadow is automatically updated

Term	Description	Ref
Digital Model	A virtual representation of a physical object that does not feature any automated data exchange with the physical object it represents.	[130]
Digital Thread	The unbroken data link through the system's lifecycle, connecting conceptual design, requirements, analysis, detail design, manufacturing, inspection, operations, refit and retirement.	[154]
Digital Shadow (DS)	An elevation of the digital model, enabled by an automatic one-way flow of data from the physical object to its virtual representation, enabling the digital shadow to reflect changes in the state of the physical object.	[130]
Digital Twin	A live digital coupling of the state of a physical asset or process to a virtual representation with a functional output.	[33]

Table 2.1: Definitions for concepts related to DTs that are differentiated by implementation.

to reflect the physical system's state but does not automatically produce output or insights. This manner of producing output is sometimes referred to as a "manual" data flow [130]. A digital shadow may not have a functional output directly connected to the physical asset, but it should have a clearly defined purpose that it fulfils [20].

In the same way that a digital shadow requires a purpose, a DT should also be designed to fulfil a particular purpose. In [33], this is described as a functional output. As an example of the lack of clarity on the subject, they define the functional output as deliverable to a system or human observer, with no requirement that it automatically be passed back to the physical asset. The functionality of the DT will depend heavily on the architecture and components of the physical system or systems that are being twinned. This is true for the highest-level components of the physical asset and its digital counterpart(s) and extends into the lower-level components related to the system's implementation. Each component's suitability needs to be considered in terms of those components that it directly interacts with and how it will enable the overall system to perform as a digital alternative to a physical entity.

To link the concepts together and delineate where the differences lie let's explore a simple example using car manufacturing. If we plan to construct a car, let's assume that we begin by creating designs for it. To better understand the properties of the car we plan to build, we can construct digital models of its components that show how the components within the car respond to the stimulus of their environment. These models are not classified as a DT or any other definition since they are not directly connected to or representative of any specific instance of our car design.

If we then begin manufacturing the designed car, we can collect data about the factory that

made it and the materials used to make the components (such as who supplied them), and we might take measurements of the individual components as they are produced. When the components are assembled into a car that leaves our production factory, we have an instance of our car. To complement that car, we also have its designs and specifications for its components as they were built. With these data sets, we have begun a digital thread for each car we're producing. Using this, we should have the data required to reproduce any car we make. As we test the car and inspect it before putting it out to market, we can add these findings to the thread. Since all of these aspects of the production are linked together, data can be shared across lifecycle stages. If defects are discovered in the car several years after manufacture, efforts to understand where they originated from and their impacts can benefit from this timeline of data. If a defect requires the car to be recalled or undergo maintenance, details of what was done should be added. If a digital thread has been well adhered to, when the car is retired at the end of its lifecycle, it should be possible to take the car apart and trace the origin of every component.

Let's suppose that during the manufacturing of our car, we embedded sensors to collect data during its operational life: engine temperatures, tyre pressures, speed, braking, etc. At any given moment, we can collect data about the car's current state, changing as the car does, like a shadow following an object. We've now constructed a digital shadow of the car. The data collected through the digital shadow is stored within the digital thread. In doing so, we've substantially enhanced the quality of the digital thread for our cars. At this point, we could use the design and manufacturing data to produce an identical car and, using a hypothetical perfect physical simulator, recreate its environment from data collected by the digital shadow to reproduce the state of that car at any given moment in its operational life. Even without this hypothetical simulation environment, we can now use this data to better understand if defects occur because of a fault in the production or through some event in the car's operation. If the car has been involved in a crash or if a fault occurs within the car, that data is collected by our digital thread. It can also help us if we use it in aggregate. If we've designed our car to operate in cold climates and find that it struggles in warmer temperatures, we can examine the data collected from the cars that are struggling by their digital shadows to help us analyse why. Since our digital shadow only collects state data and does not utilise a representation of the system to provide a functional output, this is also not a DT.

As we use our digital shadow system, we notice a consistent pattern in the process of collecting and analysing car data. Our teams repeatedly use a digital model that shows the internal components of the car's engine. When one of our cars is reported to have an engine failure, we can tune the model parameters using the properties of the materials used and the specific measurements taken during the manufacturing of that engine. This makes the digital model of the engine better reflect the state of that specific failed engine. We can then use the operational data collected through that engine's lifetime to simulate its life and identify the cause of the fault. At this point, some may consider what we have constructed to be a DT of that car's en-

Term	Description	Ref
Digital Twin Prototype (DTP)	A digital representation of a physical system to be built, containing all the required information to produce the system. (see also: "born digital" [54])	[88]
Digital Twin Instance (DTI)	A specific corresponding physical product that an individual DT remains linked to throughout the life of that physical product	[88]
Digital Twin Aggregate (DTA)	The aggregation of all the products that have been made. We can collect and aggregate the data from the population of products to provide value.	[89]
Digital Twin Environment (DTE)	An integrated, multi-domain physics application space for operating on DTs for various purposes.	[88]

Table 2.2: Definitions of DTs and related concepts.

gine. However, as our tuned model isn't directly connected to the car's engine, i.e. we have to "manually" transfer the data from the digital thread into the model when we need it, it is not yet a DT [130]. There is no "twinning" aspect since a change in the engine is not reflected in the model representation without an operator completing the loop.

To take the final step to constructing the DT, we need to combine the data and the models we have. Through our simulations, we discover that our model can identify a failing engine before a total failure occurs. We create an instance of our digital engine model for each car and parameterise that model with data taken from that each car's digital thread. As new data is collected through the digital shadow, our model can access it. In real-time, or at an appropriate interval [33], the model collects the most recent usage data and combines it with its current state to analyse the engine for signs of failure. Afterwards, it updates its internal state, stores any results to the car's digital thread and reports any faults to the owner. If we have designed this system well, then at any given moment, the state of our engine DT should reflect that of the engine in our car on the road. Through this combination of digital model, digital thread, and digital shadow, we have constructed a DT of the car's engine to detect engine failure. If we wanted to extend the functionality of our DT, we would need to enhance or integrate additional models into the system accordingly to use it for this new purpose.

Table 2.2 shows a list of types of DTs that have been proposed, each with a different purpose. Two types of DT were proposed in [88]; Digital Twin Prototype (DTP) and Digital Twin Instance (DTI). A third type, the Digital Twin Aggregate (DTA), was proposed later [89]. As seen in Table 2.2 a DTI is equivalent to the previously discussed concept of a DT, a one-to-one representation between a real-world system and a virtual representation. The DTP is a DT that predates the construction of the system being twinned. This DTP contains all the data required to construct

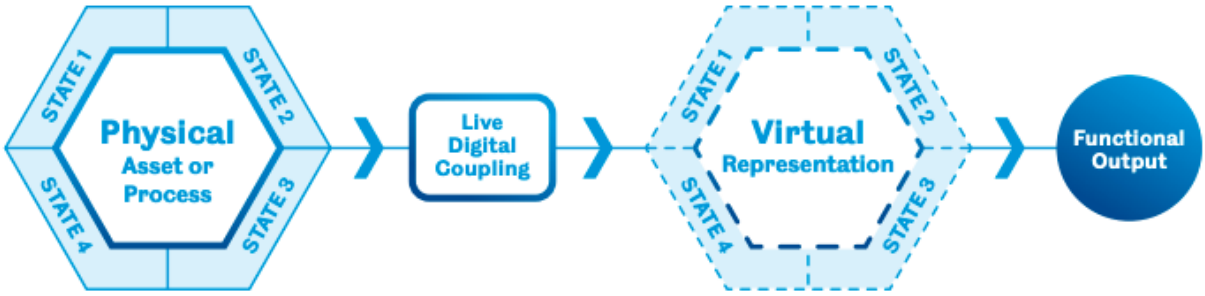


Figure 2.4: High-level components of a DT, as presented in [33]

and accurately simulate a system to be built so that physical prototyping can be drastically reduced or eliminated. To put this in the context of the car manufacturing example in this section, a DTP might be a multi-physics model of the full car design, incorporating the engine model and embedding that simulated car within a realistic simulated environment. The final type of DT proposed is the DTA. An abstraction of the DT paradigm that doesn't represent a single physical system but instead represents many combined instances to identify trends across a fleet of DTs.

As has been presented in this section, there are many definitions of DT, and they are not consistent in their attempts to specify precisely what a DT is. It is, therefore, important to summarise the understanding of a DT that we will be using in the remainder of this thesis. The purpose of a DT is to facilitate better operational management of a physical entity. In order to do this, the DT utilises an adequate representation of some aspect of the physical system or process; from this point, we shall refer to this subset of system behaviours that are of interest as the physical asset. The data required to inform decision-making is collected through sensors embedded within the physical system or its environment and is then sent at a suitably real-time rate to the DT. Within the DT, one or more models may be used individually or in parallel to compute useful outputs that can be used to better inform decision-making. These metrics are then output to a user, the physical system or another third party to assist with decisions related to the physical asset.

2.3.3 Components

The structure of a DT consists of a physical asset, a virtual representation of that asset, a real-time data connection between those two systems, and an output (as shown in Fig. 2.4). We will describe the key aspects of each of these components in this section. For a comprehensive taxonomy of the characteristics of DTs, see [224].

With the exception of in a DTA, a unique one-to-one mapping from a DT to a physical asset is required to enable the accurate modelling and replication of behaviours of the specific asset [33, 83, 224]. Sensors are used to capture the physical state of the system, comprising the asset and its environment, and enabling the recreation of this state in a virtual space [112].

The physical twin's behaviours are modelled by one or more virtual models that simulate the state of the physical twin using sensor data [112]. This state includes all of the parameters that are being measured to inform decision-making. How accurately the model represents the physical twin is dependent on the application. It has been argued that mirroring to an atomic level is required [88]. However, more realistic definitions argue that a limited "context" of the physical state needs to be measured and reproduced in the virtual space [226]. The state of a perfect DT will exactly match that of the physical system throughout its operation. In practice, all virtualisations will be somewhat inaccurate due to transmission delays and modelling uncertainties.

There are several ways that the behaviours of a physical asset can be modelled, the main ones being engineering physics models and machine learning models. A physics model performs finite-element analysis using design artefacts that specify the structure of the components in the physical system, such as Computer-Aided Design (CAD) models. By solving equations that predict the behaviours and interactions of these simpler sub-components, a representation of the full system can be constructed. The advantages of this approach are that the model design can be examined for diagnostic and evaluation purposes, and they extend well to applications outwith their original use case. For example, if the material properties of a component change, then its attributes can be adjusted to reflect this, and the updated model can resume operation immediately. However, even small models can be computationally expensive to run and increasing accuracy dramatically increases the resource requirements needed. Alternatively, ML methods can be used to train a model that learns the behaviour of the real asset from previously gathered training data. In general, ML approaches are more accurate and cheaper to operate. However, they are expensive to train, cannot be internally examined for biases and do not extrapolate well to problems not represented within their training data sets [126]. Some applications attempt a hybrid, or grey-box, approach using a physics model as a base with ML techniques to improve accuracy by accounting for discrepancies between the designed physics and those observed by the asset in the field [226].

A data link is required to connect the physical and DTs. The largest limiting factors here are connection bandwidth, speed and security. Different connections can be used depending on the use case. In ICS environments where most critical components use a wired connection, the connection to a DT will almost always be wired, except in large-scale environments that require remote facilities. Some exceptions to this could occur in instances where remote sensors are connected through wireless connections. This provides the additional security advantage of creating a closed network, reducing access points that can be targeted by third parties.

The connection between the DT and the physical twin should be live [112], to ensure that no appreciable state difference between the two occurs. Violating this requirement can potentially undermine the utility of the DT as a decision-making tool [33]. How much connection lag is acceptable will depend upon the specific scenario. In one example, previous tests on surgeons

determined that lag times between 30 ms and 150 ms would be undetectable to surgeons during the operation of a remote surgical arm [143]. However, when exposed to a Denial of Service (DoS) attack, the increased lag would cause the arm to become unfit for operation.

The key elements of the data link within the context of the Industrial Internet of Things (IIoT), IoT using industrial equipment and protocols, are shown in the Hourglass Model in Fig. 2.5. The physical and link layers respectively deal with the encoding, framing and physical transmission of data. The network layer directs data between routers across the network using Internet Protocol (IP) to get messages to the recipient's IP address. The transport layer handles the exchange of messages between processes hosted on the sending and receiving devices. Many different protocols are used at the transport layer depending upon the connection needs; the most popular two are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP provides reliability, ordering of packets, and error checking that provides retransmission of lost data. Conversely, UDP provides lower-latency connections with no error checking, meaning that any unreliability in the underlying network is not handled. The application layer is where the network connection directly integrates with the software application. At this level, various communication protocols are employed. IoT-specific protocols are primarily designed to facilitate efficient machine-to-machine (M2M) communication. However, in an ICS environment, a wide range of proprietary OT protocols are also commonly used. For a DT to effectively interact within this environment, it may need to communicate through these diverse protocols, ensuring compatibility across both IoT and OT networks. An overview of the technologies, protocols and applications can be found in [8]. A summary of the properties of the most common IoT protocols can be found in [119]. While this wide variety of protocols are useful for enabling solutions with differing requirements, they add complexity to the challenge of understanding how to secure a DT.

Finally, a DT must produce a functional output that can be used to inform decisions. The presentation of this output could be numerical or graphical, but it should aid the user in understanding the system's behaviour [112]. The human operator then completes the control loop by making decisions about the management of the asset using the data from the DT; this is referred to as a human-in-the-loop control system. It has been proposed that future iterations of DTs may use this functional output to make autonomous decisions which directly affect the physical system, whereby completing the control loop [33, 54, 226].

2.3.4 Usage

The DT paradigm has many useful applications across many different industrial sectors, including healthcare [68, 209], maritime and shipping [156], energy production and distribution [62, 245], smart manufacturing [210, 211, 248], city management, aerospace, automotive, ICS, sustainability [195] and agriculture [184]. In this section, we consider some of the DT applications that are most relevant to ICS, particularly in the sectors of energy generation, energy

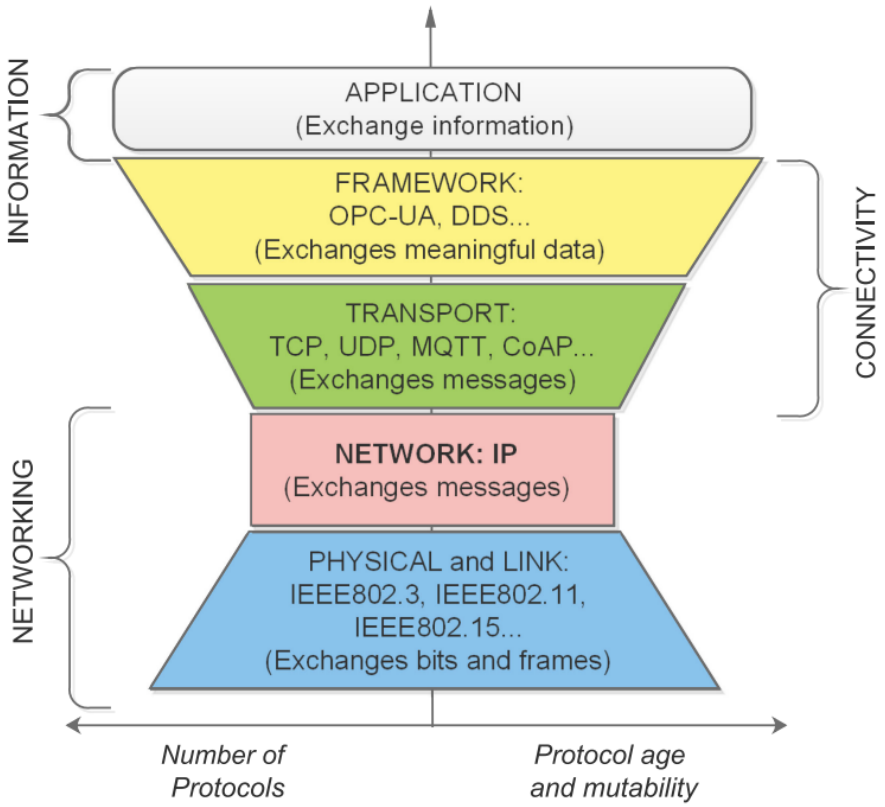


Figure 2.5: The Hourglass IIoT Stack, from [200]

distribution and smart cities. Within these domains, the main usages of DTs are in design, asset management and security.

Asset Management

Digital Twins (DTs) are increasingly used in energy sector digitization [82]. Between 2000 and 2019, global energy consumption increased by 42% (124.7 EJ), including an 80% rise in electricity usage [2]. To meet climate goals, reliance on fossil fuels must decrease, requiring a rapid expansion of renewable and nuclear energy. However, in 2019, coal and oil still accounted for nearly half of the global energy supply [2], and amid a renewable shortfall in 2022, coal production reached a record high, supplying one-third of global energy [105].

Fossil fuels offer a reliable backup when renewable sources fluctuate. While renewables are cheaper and cleaner, their dependability remains a challenge. Addressing this requires better supply forecasting and demand management across the power grid. DTs can help by integrating data from digitised energy production processes, enhancing predictive modelling, resource management, and operational efficiency.

DTs have seen applications in many energy generation sectors. In hydroelectric power, they can be used to optimise operations and enable predictive maintenance that allows for component replacements to be identified in advance and performed during scheduled downtime to maximise

efficiency [170]. While DT technology is still relatively new in nuclear energy, it is proposed to help with real-time synchronisation, state prediction and fault diagnosis, decision-making support and multi-terminal collaboration [159]. In [109], three applications of DTs to the Joint European Torus (JET) divertor were outlined. The developed DTs enable operators to prepare for, monitor and perform post-processing of data for each pulse that is performed, improving operational range, reliability and predictability when conducting experiments [109].

In wind energy, DTs been used in design, testing and monitoring. They have particular appeal in offshore environments where accessing the physical wind farm can be hazardous or expensive. Digital models of wind turbines, such as that of [148], have many uses, but there is limited evidence of deploying them on production systems using real-time data. Engineering modelling of this type helps with the design of wind farms and, if deployed in a real-time environment, can help operators to better predict the turbine's output under different weather conditions. The use of neural networks for fault detection on a simplified wind turbine model is shown in [241]. Additionally, DTs have been used for prognostic maintenance scheduling. In [201], SCADA records were combined with aerodynamic and finite element models to create virtual sensors that infer strain in areas where there are no physical sensors in order to predict the expected remaining life of components.

The utility of DTs in the field of energy distribution has also been demonstrated, often in combination with smart grids that integrate advanced monitoring processes to adapt the flow of electricity to reflect changes in supply and demand. DT implementations have been shown to help with this by using a combined approach of mathematical modelling and machine learning to detect disturbances in the flow of electricity in near real-time [220]. A different approach using In energy management, the approach presented in [84] presents a method where multiple DTs work together in a simulated environment to negotiate either providing or utilising energy within a mini-smart grid. Additionally, the authors present two types of supervisory DTs: one that attempts to learn the production and consumption demands of each node in the network and another that regulates and manages issues that occur during the operation of the smart grid.

Security

While asset design and management has been the primary focus of most DT usage, it has been observed that the DT presents opportunities to improve cyber security [55, 60, 66, 98]. DT usage to improve cyber security is a newer concept than those proposed for asset design and management. We will examine how the DT has been proposed as a means of solving cyber security challenges across the different activities routinely performed in ICS cyber defence. Of the categories within the NIST Cybersecurity Framework, DT applications are concentrated particularly towards detection, protection and identification, though they could potentially see wider usage in recovery in the future. Different features and implementations of DTs lend themselves to different security activities [60]; in this section, we'll discuss how DTs are currently being used

in cyber security.

Many current implementations of Cyber Security Digital Twins (CDTs) primarily focus on anomaly detection. By modelling a physical process, CDTs can identify deviations from expected behaviour, similar to fault detection approaches in [159, 241]. In [69], a hydroelectric power plant DT applies PLC-based invariant IDS techniques [4], using modified PLC code to collect sensor data and compare DT-predicted outputs with real values to differentiate sensor faults from measurement errors. The study also demonstrates using DT outputs as fallback sensor data in case of failures. However, anomaly detection alone is insufficient for distinguishing faults from attacks. In order to support response activities effective CDTs should develop to integrate process behaviour with control system and network activity analysis.

CDTs have also been proposed to help identify vulnerabilities in the systems that they replicate [27, 66]. This is a far less developed application of DTs and is strongly intertwined with several related concepts within the ICS cyber security space, particularly ICS testbeds, honeypots and cyber-ranges.

ICS testbeds are commonly constructed to analyse a system's security, develop new technologies and identify vulnerabilities in a safe environment with simplified representations of the full system [80, 97]. While traditionally, these would have been physical replicas of the system, the use of virtual components within testbeds has become more prevalent, such that wholly virtual testbeds exist alongside a hybrid approach which uses physical and virtual components [46]. A CDT performing vulnerability analysis would contain an entirely virtual system representation. Within the definitions of DT, if a virtual ICS testbed were constructed to represent a prototype ICS system to be built, then that testbed could be considered a DTP.

Honeypot [74] components represent components that may or may not be present within the operational system. They function as decoys of attractive targets to aid detection of attackers that attempt to interact with them [152]. While honeypots may use similar simulation or emulation approaches to represent the behaviour of an ICS component, they replicate an actual physical component in operation.

A cyber range is a simulated environment used to assess a system's susceptibility to cyber-attacks [243]. It enables red-teaming exercises [151, 240] to evaluate potential vulnerabilities. While cyber ranges and DTs share similarities, their purposes differ: a cyber range models the adversary's action space, allowing unrestricted cyber-related activities, whereas a DT mirrors system behaviour, ensuring identical outcomes between the DT and the physical system [88].

DTs share characteristics with each of those related concepts. However, their unique linkage with the physical system can make them better suited to certain tasks than these alternative, disconnected concepts, which target specific cyber security challenges.

It is proposed that a connected CDT could replicate the security characteristics of the physical system and show an awareness of cyber threats while performing other DT functions. To do this, an awareness of cyber threats and features including network architecture, hardware speci-

fication, firmware, operating system, software versions, and configuration therein is represented to some extent within the virtualisation [66]. An early proposal of this approach based upon emulated ICS components comes from [65]. They proposed that emulation of the control system, which they demonstrate using AutomationML in MiniNet, would enable security experts to test the security of the physical system without needing access to or causing risk to it. However, it could be argued that this capability can already be sufficiently satisfied through a disconnected virtual testbed, indeed the approach demonstrated in the paper results in a disconnected representation of the physical system being modelled. An example of this type of system is presented in [167] where another MiniNet-based (MiniCPS) approach is used, this time enhanced with an EPANET simulation to integrate physics simulations alongside the control system and network simulators to construct a disconnected representation of a water distribution system.

2.3.5 Threats

While DTs offer significant benefits to the systems that they twin, they also introduce security challenges in their integration [10, 98, 135, 233]. The central issue with DT implementation is the considerable increase in attack surface that could present increased vulnerabilities to the underlying system being twinned. The DT contains details about the configuration and behaviours of the twinned system, making it a high-value target for cyber security attacks. Its utility is so inherently intertwined with its ability to see into and understand the operations of the physical system that this makes it a centralised point of weakness that can be exploited to gain observation and potentially even control of the physical asset [233]. This presents challenges connecting high-performance computing solutions with the traditional legacy networks that contain vulnerable OT devices [10]. This convergence poses a significant challenge for DTs and necessitates careful consideration of the network architecture used for their deployment. These issues have led to justified concerns about the wisdom of integrating DTs into critical systems until their associated security risks have been better understood [10].

To assess the vulnerabilities of a DT, the different layers of the system must be considered. Dividing the structure of the DT into its different components allows for the threats to each part of the system to be considered. In [10], a DT system is divided into four layers and the security threats at each layer are listed. A representation of this architecture is shown in Fig. 2.6.

Several authors have attempted to classify the types of vulnerabilities that are common in digitally twinned systems [10, 55]. Thirty-four types of threats to the system were identified in [10], with half of these relating to the networking infrastructure covered by the digital thread. The data being sent to or stored within the DT provides valuable insight into the physical environment's behaviours and architecture [135]. If the system's design allows data to be intercepted, this enables attackers to identify vulnerabilities and gather reconnaissance information for future attacks. Over time, this can be used to interrupt or degrade the transmission of data to the DT, undermining the synchronisation between the physical and digital spaces. The tight opera-

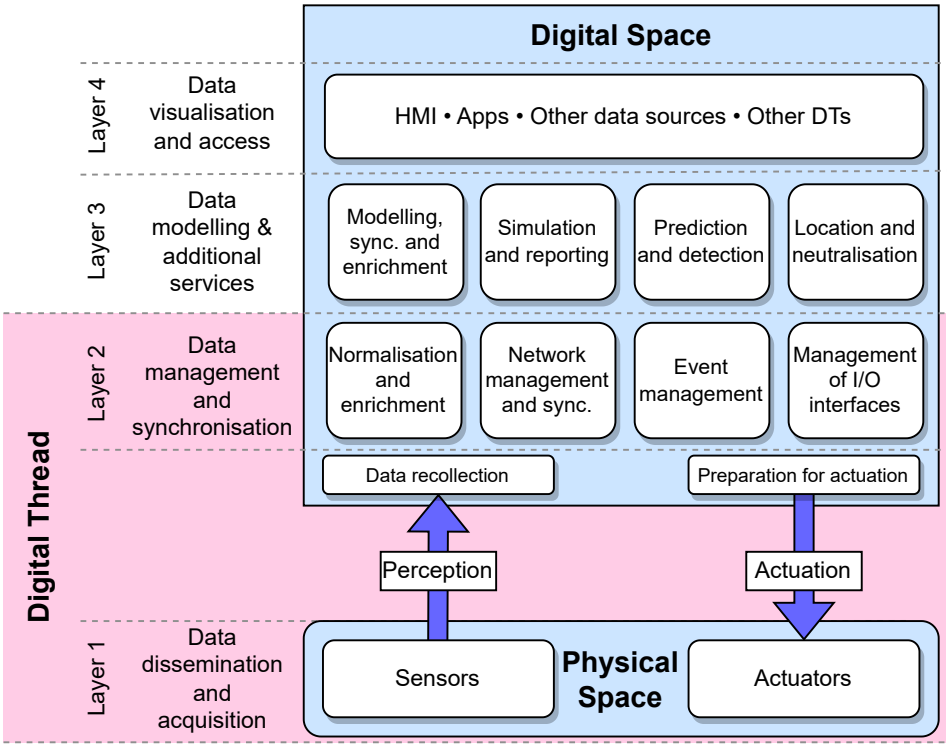


Figure 2.6: Diagram showing a four-layer-based DT as presented in [10] with examples of some of the operations performed at each layer.

tional coupling between physical and DTs leaves it particularly susceptible to attacks that target that connection bandwidth sniffing, data injection, data delay, and model corruption, which are described in a semi-formal syntax in [135].

The technologies of the digital thread encompass the first and second layers of the DT system, as shown in Fig. 2.6. In an ICS the first layer contains PLCs, RTUs and field devices. The threats at this layer largely pertain to weaknesses in the security of the networking protocols and the limited security measures deployed to the end devices. The MITRE ATT&CK ICS Matrix [49] lists 31 different techniques in this domain, each with the ability to directly or indirectly disrupt the operation of control devices that the DT relies upon to see the state of the physical process. These techniques can be used to extract private information, perform digital thread tampering, or cause physical damage [10]. Key mitigation steps are to follow the guidance discussed in Section 2.2.3, ensuring a secure segmented network design and developing secure means of providing data to DT data collection systems such as historians.

Layer two systems handle most of the data management and synchronisation of the system. This is where the data is organised to present a clear picture of the built environment to the DT and its operators. Meanwhile, tasks such as storage and normalisation are performed on the collected data to prepare it for use in modelling and simulation. Attacks at layer two attempt to retrieve data from storage in the DT via software bugs, privilege escalation, deployment of rogue DT servers or man-in-the-middle attacks [50]. They may also attempt to use exploits

or supply chain attacks to modify the function of the DT server [10]. These technologies are largely complex, layered IT-based solutions. Servers may contain multiple hosted processes that interact with devices across the network, enabling an attacker to spread through a system across components from different subsystems [50].

2.4 Formal Methods For Cyber Security

With the increasing prevalence of cyber security threats, a secure design has become an almost ubiquitous requirement for most systems. However, providing assurances that this requirement has been satisfied is a challenging task. Formal methods are a set of mathematically rigorous tools and techniques that allow us to formally specify systems and verify their design with respect to their requirements. By developing structured approaches to represent cybersecurity components and requirements, formal methods can provide increased assurance that a system's design meets its security goals.

Formal methods build upon a long history of applying mathematical analysis to assess the soundness of a design. However, their application in the far more recent concept of cyber security started in 1990 when Burrows *et al.* published a formal logic that used a refinement model checker to detect a new triangular attack in the Needham-Schroeder protocol [31]. Since then, formal methods have seen widespread use in protocol analysis [139, 169], system certification [47, 207], hardware verification and application design.

Since security vulnerabilities must be specified before they can be identified through formal verification techniques, the user must know what they are looking for before a vulnerability can be found [134]. Formal verification approaches are therefore best applied to identify known vulnerabilities. In this respect, applying formal methods to new technology, such as DTs, presents challenges as the formal specification of the system depends upon the system architecture, which has yet to reach a state of maturity. Evidence for this fact can be found in the relatively small number of publications modelling DTs with formal methods in any context. However, if the system can be broken down into components which are well understood, such as the architecture of an ICS, and threats that are well documented, then formal method approaches can be applied towards identifying known threats on these component systems and how they might interact with and impact other aspects of the system.

2.4.1 System Specification

Model checking is the most common formal method to be used in the context of security analysis of industrial systems [134], spanning the domains of vulnerability analysis, protocol analysis, and network security.

Formal methods have been applied to the domain of verifying the code of ICS systems. Verification of PLC code has been performed using a number of model checking approaches.

Graphical PLC languages do not lend themselves to direct formal analysis, so an intermediate step is often used. In [250] an approach is proposed to check for malicious code injections in PLCs. This is done by translating Instruction List (IL) code into an intermediate language based on Vine IL [203]. The resulting representation can then be used to generate temporal execution graphs that can be used to check if the code reaches a list of safety-violating states. PLCVerif [52,215] is a tool for analysing PLC code written in Statement List (STL) or Structured Control Language (SCL) or translated through Instruction List (IL) [53], with model checkers; currently NuSMV, nuXmv [32], Theta [221] and CBMC [131] are supported. The use of the tool is described in [71] where it is applied to detect bugs in PLC control systems at CERN. NuSMV [38] has been used separately to model the design of a temperature control system in [199], and to model a system based on a robotic arm controller in [128]. The models developed in both cases could be used to identify design flaws that would cause the underlying system to malfunction if exploited. Several approaches use timed automata to represent the time-changing state of the PLC. The timed-automata analysis tool UPPAAL [147] was used in [229] to verify PLC code, and the Process Analysis Toolkit (PAT) was used in [230] to test the resilience of a system with an automatic recovery mechanism.

Modelling physical processes that contain continuous behaviours using formal methods is a challenging task as it there are too many possible states. Modelling these processes, therefore, requires a discretisation step to ensure that the representations of the system can remain searchable in a reasonable timeframe. Discretisation involves the grouping of continuous measurements or intervals of time into steps that can be more easily differentiated within the model. An example can be seen in the representation of a water treatment process in [189]. The ASLan++ [225] tool is used to search for attacks that change the state of the testbed. The state of different system components is grouped into intervals such that the water level in a tank may be represented by the value “high” rather than a numerical value. When combined with discrete time steps, this can represent the effect of a pump being turned off incorrectly and causing the system to enter a dangerous state, such as an overflow. In [117], a similar discretisation approach in which PLC code is represented in Alloy [114] is used to analyse the performance of the same testbed as [189]. In [160], the authors construct a model of a SCADA network in a water distribution system as a network of timed automata in UPPAAL. The model is derived by discretising a dataset of system logs, and is then evaluated against a suite of 10 attacks.

A security threat occurs when a malicious actor can exploit a vulnerability to cause an effect. The approaches listed so far secure CPSs by identifying these vulnerabilities. They feed into a broader movement towards developing secure coding practices in industrial control environments to increase the robustness of these systems [1]. However, in an industrial control environment, some code may need to adopt less robust code practices to ensure a safe operating environment or to support operators in the operation and maintenance of the system. While

[1] https://plc-security.com/

vulnerability analysis can help increase the robustness of system code, some vulnerabilities may be intrinsic to the system's needs or may stem from hardware vulnerabilities in the systems on which the code is run [13, 161, 234]. To address this issue, several other formal approaches have attempted to assess the network security of the system to assess how these vulnerabilities may be exploited.

There have been several different approaches for modelling the cyber security of IoT and CPS networks. A recent survey reviewed over eighty specification languages and verification tools used for the analysis of cyber security methods [134]. The approaches surveyed were categorised into those using model checking [19, 90, 168, 189], theorem proving [231] and lightweight formal methods [113], each of which can be applied at different levels of security analysis.

An initial framework for cyber security verification of IoT systems using the Alloy Analyzer is presented in [138]. An IoT architecture is modelled as a collection of segregated subsystems consisting of sensors and actuators communicating data over network channels. Attack patterns are presented for identity-faking and eavesdropping attacks, which are then modelled in Alloy using a concept of malicious data and compromised systems. Mitigation strategies are presented, specified and verified as part of the approach. This is then extended in [136] to demonstrate how the set of devices modelled can be expanded. This enables the analysis of data packet tampering and brute force attacks along with their mitigations, as specified in the ISA/IEC-62443-3-3 standard [107].

In [155], a middleware framework for facilitating security requirements discussion between development engineers and security experts within the automotive domain is proposed. The framework uses a UML specification of the system that is then translated into a representation in the Alloy specification language [114] to identify threats from a defined list of known attack patterns. The approach uses a combination of components, channels, messages and interfaces to represent an autonomous emergency braking system. Instances consist of a single state with message transfer occurring instantaneously between components. The analysis results can then be used to identify vulnerabilities and promote discussions about the risks being accepted in the design of a system, proving the value of utilising formal methods in a context where security guarantees cannot be given.

Specifying cyber security vulnerabilities in formal methods approaches often requires constructing a representation of a threat actor, known as an attacker model [188]. In formal analysis of a system, it is often best to consider the most extreme case so as to fully examine the properties of a design, so an attacker model should fully characterise the possible interactions of the attacker and the system under attack [188]. The Dolev-Yao model [63] is a threat model originally used in a security analysis of protocols. It models an attacker as an adversary that has full control over the communication layer of the network, enabling them to overhear, intercept and synthesise any message. This is commonly described as "the attacker carries the message".

This approach is well suited to protocol analysis as it strips away all other security sources, focusing the analysis on only the basic cryptographic elements of the protocol being analysed. This attacker model has also been widely used in information security, with increasing adoption in the domain of CPS [56, 189]. However, attacker models can be integrated into the model in different ways, depending upon the nature of the system being modelled, and may have different characteristics (termed as "dimensions" in [188]) to focus on different aspects of system security. Since a Dolev-Yao model represents an attacker that has full knowledge of and access to the system network, it will always represent an insider attack, where an authorised user interacts with the system maliciously. However, other models focus on other types of outsider threats that are distinguished by their different resources, determination and how they prioritise the need for stealthiness to present varying dimensions of the attacker profile.

TLA+ [144] is an exhaustively testable pseudocode and was used in [137] to model a cloud-connected engineering terminal within an ICS environment. The system is represented by the set of actions that each of the systems can perform. Attack scenarios for malicious firmware attacks and a brute force authorisation attack are then examined, and a proposed mitigation strategy for each is checked. A similar system is considered in [216], where it was demonstrated that VDM-SL [28, 29] can be used to analyse firmware signature checks and user authorisation processes that were then searched for the presence of malicious firmware attacks and expired and leaked token attacks.

2.4.2 Intrusion Detection

To assess the effectiveness of an IDS, its classifications are evaluated against a known ground truth. The primary evaluation metrics include false positives (incorrectly flagging benign activity as an intrusion) and false negatives (failing to detect an actual intrusion). Alongside true positives (correctly detected intrusions) and true negatives (correctly classified benign activity), these values are used to compute key performance indicators such as accuracy and F-score [34, 212].

The use of model checking for intrusion detection is less common than other approaches. There are three main categories of intrusion detection methods: anomaly detection, misuse detection, and specification detection [172]. Specification-based approaches are considered to be the most reliable of the three categories [223]. This is because they provide fewer false negatives than anomaly detection techniques, while still being able to detect novel attacks, unlike misuse detection methods. Model checking, by its nature, is a specification-based approach using a specification of the system to generate a model of the system. This allows for a comprehensive representation of all the possible system behaviours from which deviations can be detected. However, this approach still requires some level of abstraction in order for the model checker to be able to analyse the model within an acceptable period of time.

A state machine representation of system design specification has been proposed as a means of identifying intrusions in a safety-critical medical environment using a peer-to-peer monitoring

approach [165]. The approach uses specification data to define behaviour rules for devices and uses threat models to differentiate between attacks and software bugs, thereby reducing the rate of false positives. They demonstrated the viability of their methodology on a probabilistic state-machine representation of a medical case study using behaviour rules derived from specification data.

In [90], a method of applying software validation through model checking is used to detect modifications to PLC code. The approach leveraged previous techniques to translate PLC code into UPPAAL representations [5,52]. The UPPAAL models were then used to detect changes by comparing these representations to a trusted ground truth PLC program. This type of approach creates a more specific version of a program hash of the code, and can be useful since PLCs broadly do not have the computational resources to execute program hashes [235]. However, if the attack targets the network outside the PLC, these methods will not be able to detect it.

Specification-based approaches have also been developed for monitoring network traffic [196, 204, 217]. They've been applied to low-level TCP/IP networks [196], ad-hoc mobile networks [204,219], voice-over-IP protocols [217], smart grids [23] and ICS [101]. In [101], safety and security standards are used as the basis for deriving LTL properties that are used to detect system anomalies. These approaches perform well in environments with a small number of protocols. However, they can face scalability issues in environments with a larger number of interoperating protocols [23] which are in part related to the need for expert knowledge during the specification-extraction process [101].

2.5 Summary

In this chapter, we have presented a summary of the background related to the research presented in this thesis.

We began with a review of ICS, contextualising their current cyber security challenges through the lens of their historical evolution. We outlined the heirachical structure that these systems ahere too and described the devices, networks and protocols that are used at each level of their operation and supervision. We gave examples of prominent attacks that have demonstrated the vulnerability of these systems and that hint at the potential impact of future attacks. We then outlined the cyber defence initiatives and guidance that are working to meet this rising cyber threat.

We then introduced the concept of a DT as a virtual representation of a physical asset or system. We described the origins of this concept as a PLM tool that is now seeing application towards other system management challenges. We presented the recent published definitions of DTs and examine their characteristics through explanations of DT components. We outlined some usages of DTs for asset management and security, particularly in ICS environments used in energy generation and transmission. Finally, we examined the structure of the DT for cyber

security vulnerabilities and identified the digital thread that connects the DT's virtual representation to the underlying physical asset as a target for cyber attacks. We described how these attacks can undermine the the ability of the DT to perform its function and therefore negatively impact the operation and management of the system that the DT represents.

We concluded this chapter with a review of how formal methods can be used to verify the safety and security of system design. In particular, we reviewed how model checking and related formal approaches can be used to evaluate ICS security. We explained how the mathematically rigorous nature of these approaches enables them to provide strong evidence of the security of the systems they analyse than traditional testing methods, provided that the underlying specification used in the analysis is well-constructed. We provide examples of their use in protocol analysis, validation of PLC code, and network modelling in IoT and CPS environments. Finally, we identified several methods through which formal methods have been used in specification-based intrusion detection approaches to reliably detect system anomalies while demonstrating low false positive rates.

Chapter 3

Preliminaries: Formal Methods

In this chapter, we introduce the formal modelling concepts and techniques used in the application of formal methods. In particular we will identify model checking approaches and how they have been previously applied to create formal representations of computer networks in the past. We also consider approaches using formal modelling techniques for anomaly detection in ICS. Having identified SPIN and Alloy as the two techniques that best meet our requirements we will give an introduction to their specification languages and the methods of performing model checking with these tools. By the end of this chapter, we will have discussed the formal modelling background required for this thesis and introduced the concepts, formalisms, tools and techniques required to understand the formal methods used in this thesis.

3.1 Formal Methods

When the design of a system or process becomes complex enough, it can be hard to fully understand the breadth of behaviours that can occur in the designed system. As systems have become more complex and reliant on hardware and software, the consequences of failures have also intensified; in the most severe cases, failures can result in significant financial losses for owners [64, 91, 125] and the loss of life for users [45, 242]. This problem is exacerbated in the current cybersecurity landscape, with threat actors intentionally seeking out and exploiting vulnerabilities in software design. This rise has led to an increased interest in the application of formal methods in the safety-critical contexts [177].

Formal methods techniques use mathematical modelling to encode and analyse systems, giving insights into their design. The strength of this approach is two-fold. Modelling a system's design is an intensive process, providing a deeper understanding of the system at hand and often uncovering design flaws during the modelling process. The output of the modelling process is a mathematical representation of the system, an artefact that can be rigorously analysed for consistency and the presence of undesirable behaviour [43]. The assurances provided by this analysis are given with respect to a *property* that uses a logical expression to represent the

behaviour that is being searched for. Using this property, the formal method can identify whether or not the specified behaviour occurs within the representation of the system [132]. This process can be applied to verify that the modelled system meets its requirements or to provide tangible evidence that it does not.

There are many different methods of providing formal verification for a system, each with its strengths and weaknesses [12]. No single approach is best suited to all verification tasks. Performing mathematically rigorous verification of a system is a computationally intensive task requiring that the formalisms used to model the systems strike a strict balance between expressive power and computational efficiency. Therefore, the design of each formalism is often targeted towards specific types of systems or problems. While this thesis focuses on model-based approaches, there are other methods of performing formal verification. Theorem proving is an approach that uses deductive reasoning instead of algorithmic search to prove or disprove the correctness of a system [24, 103]. Program derivation is the process of deriving executable code from a formal specification through mathematical rules [59, 181]. Static analysis verification doesn't require a specification language and instead directly analyses the executable code either in source form or after compilation to determine its correctness without requiring execution [1, 95, 166].

3.2 Model Checking

Model checking is a technique in which a model is verified by algorithmically searching its state space. Model checking is commonly applied in software verification [22, 25, 100], including industrial domains [18, 178] and operating systems. In critical systems, it has been used to verify safety [92, 153], reliability [180, 215, 227] and security [134]. Model checking is particularly suited to identifying edge cases which may not be found through alternative testing [35, 44].

A model is defined using a specification language that is then used to construct a mathematical structure upon which analysis is performed. Specification languages are defined formalisms that describe the components of a system and how they interact. The strength of the analysis depends upon the specification model and how well it represents the system being inspected. Therefore, care must be taken to ensure that it adequately represents the system's behaviours with respect to the properties being searched for. While ideally, the model of a system would represent every component and every system behaviour, even in relatively simple systems involving a small number of interacting components, the number of distinct, reachable states that a system could reach can be vast. In such instances, while the model may be valid and theoretically searchable, in practice the time taken to search this "state space" can become infeasible given the computational resources and time available. This tendency for the size of seemingly simple models to exponentially increase in complexity to the point where they cannot be practically analysed is referred to as the "state space explosion" problem [40]. The application of model

checking, therefore, becomes an exercise in identifying and representing only those aspects that are critical to the analysis of the behaviour under scrutiny while abstracting less relevant design aspects. Some techniques, such as symmetry reduction [42, 163] and induction [41, 162], can also help to mitigate state-space explosion.

3.2.1 Model Checking Tools

The highly focused nature of model checking analysis incentivises the creation of tools that analyse specific types of problems. These different approaches are then supported by formalisms and structures that are suited to modelling system behaviours relevant to those specific problems. Model-checking tools may support qualitative or quantitative analysis. Quantitative analysis evaluates the model with respect to the value of variables within the model. This promotes reachability analysis [76] or the calculation of the probability of a sequence of events occurring [141]. Our approach uses qualitative analysis, which instead expresses undesirable behaviour through the model's semantic elements. The result is a returned counter-example showing a system state [114], or sequence of actions that broke the analysed property [99].

As with all programming languages, within these broad categories of language and tools, different specification languages focus on addressing different problems. Model checking tools typically consist of a specification language and an analysis tool for analysing models written in the language. In this section, we introduce several widely used model checkers and highlight their key differences. We then discuss the motivation for choosing the specific tools used in our methods, before we provide a more detailed introduction to them in the next section.

SPIN

The Simple Promela Interpreter (SPIN) [99] model checker is a widely used open-source software verification tool designed for the formal verification of concurrent systems such as multi-threaded software applications. It leverages a high-level, state-based description language called Promela [81] (PROcess MEta LAnguage), which is loosely based on Dijkstra's guarded command language [61]. Promela is commonly used for modeling concurrent processes and their interactions. SPIN can then verify these models by checking whether the modelled system satisfies specific properties expressed using LTL, which captures the ordering of events and their correctness over time. SPIN works by verifying that no execution path exists that violates the specified property. If such a path exists, it is referred to as a counter-example, and the property is deemed violated. Depending on the type of property being verified, SPIN searches for different kinds of counter-examples. For safety properties, where it is being checked that something bad will never happen, a counter-example is a path containing at least one state where the bad condition occurs. In the case of liveness properties (i.e. a good thing will eventually happen), a counter-example is instead a repeating cycle in which the expected good event never occurs. If

no counter-example can be found, the property is considered verified.

SPIN has been used extensively to verify industrial software systems and in the analysis of protocol design. In industrial and critical systems, it has been used to detect deadlock states in robotic software in a car welding station [237], and has identified errors in the design of flight control systems [236], including the NASA DS1 demonstrator [85]. It has also seen applications verifying system security, particularly network protocol security. Since an early demonstration where SPIN modelling identified cryptographic weaknesses in the Needham Schroeder Public Key Protocol [157], other case studies have analysed internet payment systems [213, 247], and network authentication protocols [93], including the IKEv2 key exchange protocol used in IPSec [247]. It has also been applied to IoT networking, such as in [246], where it was used to model the components of a publisher-subscriber middleware architecture. Similarly, the publish-subscribe model of the IoT protocol MQTT was modelled in [15] to validate the behaviour of different quality of service levels. SPIN was later used to verify the absence of deadlock in an MQTT variant for communicating vehicles, MQTT-CV, that results in less network traffic for subscribers [36].

UPPAAL

UPPAAL [21, 147] is a tool for modelling real-time systems. It uses a subset of systems as a network of timed automata. A timed automaton is a finite state machine that has been extended with clock variables to support the modelling of time-bounded behaviours. The clocks in the automata advance at the same rate, and actions can be synchronised, enabling multiple automata to wait to perform actions at the same time. UPPAAL can be used to check properties across a network of timed automata using a subset of Timed Computation Tree Logic (TCTL) [14, 94], an extension of CTL logic that adds clocks. UPPAAL has been widely used to test real-time systems [96], validation of PLC controller code [229] and has been used in the generation and verification of robotic operating systems [228]. In cyber security it has been shown to support intrusion detection through software validation [90] and has been used to analyse the robustness of fallback control systems in ICS [192].

PRISM

PRISM [141] is a probabilistic model checker that can be used to analyse systems that exhibit non-deterministic or probabilistic behaviour. It supports modelling of probabilities by representing systems as Discrete-Time Markov Chains (DTMCs), Continuous-Time Markov Chains (CTMCs) or Markov Decision Processes. In these modelling environments, each transition can be accompanied by a probability of that transition being taken, enabling the model checker to analyse the probability of an overall specified outcome occurring. This is a unique characteristic of PRISM and enables it to provide quantitative evidence for the fairness and performance of randomised distributed algorithms [72, 140, 142], probabilistic security protocols [146, 171] and

game design [121, 122]. It has also been demonstrated as a tool for the generation of mathematically optimal UAV controllers in a simulated search scenario [75].

NuSMV

NuSMV [39] is a symbolic model checker based upon Binary Decision Diagrams (BDDs) [7]. Systems are specified as Kripke structures to represent finite state machines. Each process can be specified as a module consisting of variables using basic datatypes that can be modified during transitions. NuSMV supports checking properties using CTL and indirectly supports LTL properties by reduction to CTL. It has been used for verification and validation in industrial applications, including in self-driving cars [77] and ICS [123], where it has been used to verify the safety of PLC programs [129, 199].

Alloy

Alloy [115] is a modelling tool consisting of a lightweight declarative programming language and an "analyzer" that analyses the generated instances. The Alloy specification language contains signatures declaring the types of elements that exist in a model, then uses first-order logic to describe the relationships between these elements. The tool was originally intended as a method of validating software design by specifying the modules and data structures used in the software. Once specified, the Alloy Analyzer returns instances with these components in different configurations. These instances can then be checked using predicates that specify invalid arrangements or sequences of behaviours. Due to its first-order logic foundation, the Alloy specification language is highly expressive and extensible, enabling it to be applied to many different domains. As a result, Alloy has seen usage verifying OS security models [57], cross-domain access control policies [187], healthcare workflows [232] and the OAuth 2.0 authorisation protocol [179]. In network security, it has also been used to model cloud-connected CPS [216], develop modelling frameworks for IoT security [138] and model man-in-the-middle attacks in multi-channel wireless networks [9].

Motivation for Using SPIN and Alloy

The identified model checkers have differing strengths and weaknesses that enable them to be applied towards developing a formal methods-based intrusion detection approach. We initially considered the available property specification logics. Since we are primarily identifying reachable states and do not require the ability to specify statements across all paths, it was determined that either LTL or CTL could be used, with LTL providing a less restrictive option for property specification. UPPAAL is well-suited to the domains of real-time systems and has demonstrated application in ICS environments. However, it is less suited to the domain of concurrent systems, and for our purposes, we are primarily interested in the sequences of data exchanges instead of

the time taken to perform them. The desire to examine the outcomes of the concurrent operation of ICS and DT components motivated the decision to pursue the use of Promela and SPIN instead. While PRISM is useful in the analysis of random and probabilistic environments, the ICS environments that we are modelling do not use probabilities in its design. As a result, we would either not be utilising PRISM to its strengths or be required to include arbitrary probabilities within our models where none exist. NuSMV also provides a viable method of specifying a network of finite state machines that can be applied to represent ICS environments and presents the option to use CTL properties. However, the decision to use Promela/SPIN instead was made based upon the more expansive data type system of Promela/SPIN, combined with the ability to incorporate synchronous and asynchronous channels for interprocess communication.

In summary, SPIN allows us to represent the different interacting components of the ICS and its supervisory environment as concurrently operating processes. These processes can exchange data, including custom C structs if needed, either synchronously or asynchronously through natively supported channels. When performing verification, we can construct LTL properties that capture the required behaviours about the existence of a path that satisfies the given atomic propositions. Should one exist, this is returned as a counterexample that demonstrates the violating behaviour.

When modelling network security, the expressiveness and extensibility of the Alloy specification language provide a strong ability to specify a large number of components within a hierarchy of inheritable properties. Alloy's declarative language and focus on constraint satisfaction separate it from the list of other candidate model checkers, enabling us to examine a wide variety of network configurations and attack sequences. The recent addition of temporal connectives in Alloy 6 increases the expressive power of properties that can be specified. Finally, the identified attacks can be visualised within the analysis environment in a manner that makes it easy to recognise how vulnerabilities are being exploited.

3.3 The SPIN Model Checker

In this section, we provide a more in-depth introduction to SPIN and its specification language, Promela. We begin with an overview of the Promela language and then introduce and example to illustrate the use of the SPIN model checker.

The core aspects of Promela modelling are:

- Promela specifications consist of process templates called *proctypes*, global message channels and other global variables.
- Each component type is declared within a `proctype` structure which contains local variables and statements.
- Each proctype instantiation represents the behaviour of an individual component.

- Guards (statements which may block, like boolean statements or channel operations) and choice statements control execution flow.
- Inter-component communication takes place via shared variables and channels.
- Two choice constructs (**if...fi** and **do...od** statements) allow us to express non-determinism in our model.
- Directly supported variable types include **bit**, **byte**, **short**, **int**, **array** and an enumerated type **mtype**. Further data types can be indirectly supported through the C compiler.
- Initialisation information - i.e. how many and which instantiations of each proctype to create when the program is run. Processes are either initiated via a defined **init** process or, via the **active** keyword.

Note that statements can be placed within an atomic block. This ensures that the statements are executed without interruption from another process. The atomicity can be broken, though, if an atomic block contains blocking statements (boolean statements (conditions) or channel operations).

To provide a concrete description of the usage of these elements, we present a grammar containing the subset of Promela constructs used in our models, in Fig. 3.1. A full grammar is available online [1].

Consider the simple Promela example shown in Fig. 3.2. There are two processes, defined as instances of proctypes A and B via the **init** process. The **atomic** clause here ensures that they both start executing at the same time. We shall refer to the processes themselves as A and B for simplicity. There is a single (global) variable, `variable_1`, which is incremented by process A and decremented by process B. Even this very simple example has several associated behaviours. Both processes are enabled in the initial state as the value of `variable_1` is 1. If process A executes two steps in succession (evaluating the guard and then incrementing `variable_1`), process B is then blocked as its guard no longer holds. Similarly, process B can execute both of its steps and cause A to be blocked. Finally, A and B can each execute their initial steps (in either order), and then they can each execute their remaining steps (in either order). There are said to be six paths resulting from this specification.

SPIN supports the specification of properties in LTL [183]. LTL properties consist of a finite set of atomic propositions, boolean connectives, and temporal operators, e.g. <> (“eventually") and [] (“always"). We do not provide details here, except to describe any property that we use in this thesis.

For the simple example in Fig. 3.2, we might (incorrectly) assume that for all paths, whenever `variable_1` is equal to 2 then it will eventually become equal to 1 again. In our example,

[1] https://spinroot.com/spin/Man/grammar.html

```
spec        ::= module*
module      ::= proctype | init | declLst | ltlDecl
proctype    ::= active PROCTYPE name ( ) { sequence }
init        ::= INIT [ priority ] { sequence }
declLst     ::= oneDecl [; oneDecl]*
oneDecl     ::= typename name [= const] [, name [= const]]*
typename    ::= BIT | BOOL | BYTE | SHORT | INT | MTYPE | CHAN
active      ::= ACTIVE
sequence    ::= step [; step]*
step        ::= stmnt
              | label : stmnt
label       ::= name
stmnt       ::= IF options FI
              | GOTO name
              | D_STEP { sequence }
              | ATOMIC { sequence }
              | assign
              | expr
              | SKIP
assign      ::= name = expr
              | name = name binarOp const
              | name = name binarOp name
options     ::= :: expr '->' stmnt [:: expr '->' stmnt]*
expr        ::= name binarOp const
              | name binarOp name
              | const
              | name
              | expr andOr expr
              | (expr)
              | '!' expr
binarOp     ::= '==' | '!=' | '<' | '>' | '<=' | '>=' | '+' | '-'
andOr       ::= '&&' | '||'
const       ::= number | TRUE | FALSE
ltlDecl     ::= LTL name { ltl }
ltl         ::= opd
              | (ltl)
              | ltl binop ltl
              | unop ltl
opd         ::= name | const | expr
binop       ::= '&&' | '||' | '->' | '<->' | 'U' | 'R'
unop        ::= '!' |    []    |    <>
```

Figure 3.1: Reference grammar defining the Promela language components used in our models.

```
1  byte variable_1 = 1;
2  proctype A(){
3      (variable_1 == 1) -> variable_1 = variable_1 + 1
4  }
5  proctype B(){
6      (variable_1 == 1) -> variable_1 = variable_1 - 1
7  }
8  init{
9      atomic{run A(); run B()}
10 }
```

Figure 3.2: Promela code showing a simple example with two processes and one global variable.

this property is stated as [] r -> <> t where `r` and `t` are defined as `variable_1==2` and `variable_1==1` respectively (note that “for all paths" is implicit for all LTL properties).

When a Promela specification is compiled and run with SPIN, a global automaton consisting of all of the initiated components is constructed and combined with an automaton capturing any included LTL property. This merged automaton is referred to as the underlying state-space of the model. By searching the state-space, SPIN can capture any executions of the model that violate the property. For the example in Fig. 3.2 and the property [] r -> <> t described above, SPIN would indicate an error and return the first path for which the property was violated. The path where A executes its two steps and B is blocked would be such a path.

3.4 Alloy

We now provide an introduction to Alloy modelling. Table 3.1 shows common terms used in the specification of Alloy models and their analysis with the Alloy Analyzer. To illustrate these constructs, we provide an introductory example, taken from the Alloy tutorial pages[2].

To further enhance the understanding of the Alloy language, we present a subset of the specification language’s grammar in Fig. 3.3. A full grammar is available online [3].

We examine the design of a file system. The system consists of a group of “file system objects” representing all of the files and directories in the system. Each object is associated with a parent object. If an object is a directory, it knows what its contents are. Finally, the system has a “root directory” at the top of the file system.

Converting these requirements into an alloy model representing the system gives the listing shown in Fig. 3.4. The model contains many of the basic constructs of a typical alloy model. Line 2 defines the basic FSObject as an element with an optional parent relation to a single directory. We define it as an **`abstract`** signature since we are using it as a supertype within

[2]https://alloytools.org/tutorials/online/frame-FS-0.html
[3]https://alloytools.org/spec.html

Term	Definition
Signature	A language construct used to describe a type of element that exists within the model.
Relation	A feature of a signature that describes how signatures are related
Instance	The output of the Alloy Analyzer when given an Alloy model written in the Alloy Specification Language. Shows a single example of the model.
Atom	An instance of a signature.
Scope	An input provided to the Alloy Analyzer detailing the maximum number of atoms that can be assigned to each signature when searching for specification instances.
Fact	A restriction on how atoms can be assigned within an instance.
Signature Fact	A restriction on how atoms of a specific signature can be assigned within an instance.
Predicate	A true statement that must be satisfied by any generated specification instance.

Table 3.1: A list of terms that are used when describing Alloy models.

which to specify relations that all objects will have, without allowing any instances of that signature to exist in the model. Directories are then specified on line 5 as a subtype of the FSObject signature, with each directory containing "contents" relations to a set of FSObjects. The **`set`** operator declares that a relation applies to 0 or more elements – so a directory may be empty. Files are declared similarly on line 8, without containing any relations. One important consideration is that all signature subtypes are disjoint subsets within the set of their supertype. The relations and facts affecting FSObject apply to all the subtypes of that signature. Finally, we specify a Root signature as a subtype of the Dir signature. This is a special case of the Dir signature that has no parent, and therefore, we require that exactly one such directory exists. This is specified by marking the signature with a **`one`** multiplicity constraint. We take this opportunity to explain the existence of two other multiplicity constraints that are used in further models but are not demonstrated in this model. The **`some`** constraint is an existence restriction, specifying one or more instances of a signature or relation existing. The **`lone`** constraint is similar to the one constraint but allows for non-existence, it specifies that either one instance or zero instances of a signature or relation exist.

Returning to our model, if we ask the Alloy Analyzer to generate instances of our model, it will show us examples that satisfy the constraints we've specified so far. One example is shown in Fig. 3.5. However, something is wrong, we have a directory that is parenting itself. We have a file that is the parent of the root directory, and we have no content relations. So, along with the requirements specified so far, a few more are required. These could be considered implicit requirements, such as specifying that all file system objects are connected and that a directory is the parent of its contents. These must be explicitly stated to represent the system correctly, which can often be what leads to the uncovering of incorrect assumptions. We, therefore, use statements specified within a **`fact`** to constrain how the relations between atoms can be formed

```
import      ::= OPEN qualName [[qualName,+]] [as name]
paragraph   ::= sigDecl | factDecl | predDecl
    | assertDecl | cmdDecl
sigDecl     ::= [var] [abstract] [mult] SIG name,+ [sigExt]
    { fieldDecl,* } [block]
sigExt      ::= EXTENDS qualName | in qualName [+ qualName]*
mult        ::= LONE | SOME | ONE
fieldDecl   ::= [var] decl
decl        ::= [disj] name,+ : [disj] expr
factDecl    ::= FACT [name] block
predDecl    ::= pred [qualName .] name [paraDecls] block
paraDecls   ::= ( decl,* ) | [ decl,* ]
cmdDecl     ::= [name :] ( RUN | CHECK )
    ( qualName | block ) [scope]
scope       ::= for number [but typescope,+] | FOR typescope,+
typescope   ::= [exactly] number qualName
expr        ::= const | qualName | @name | THIS
    | unOp expr | expr binOp expr | expr arrowOp expr
    | expr [ expr,* ] | expr [! | not] compareOp expr
    | expr ( => | IMPLIES ) expr ELSE expr
    | quant decl,+ blockOrBar
    | { decl,+ blockOrBar } | expr '
    | ( expr ) | block
const       ::= [-] number | NONE | UNIV | IDEN
unOp        ::= ! | NOT | NO | mult | set | # | ~ | * | ^
    | ALWAYS | EVENTUALLY | AFTER | BEFORE | HISTORICALLY | ONCE
binOp       ::= OR | AND | IFF | IMPLIES | & | + | - | .
    | UNTIL | RELEASES | SINCE | TRIGGERED | ;
arrowOp     ::= [mult | set] -> [mult | set]
compareOp   ::= IN | = | < | > | =< | >=
block       ::= { expr* }
blockOrBar  ::= block | bar expr
bar         ::= |
quant       ::= ALL | NO | sum | mult
qualName    ::= [this/] ( name / )* name
```

Figure 3.3: Reference grammar defining the Alloy language components used in our models.

in our model. We make the required additions in Fig. 3.6. All statements made using facts are always true in the instances generated by the Analyzer. On line 2, we explicitly constrain all directories and their contents, such that there is always a parent directory to a directory from its contents. Line 5 constrains that all FSObjects are directly or indirectly connected to the Root directory through the reflexive, transitive closure, $\star$, of the contents relation. The reflexive transitive closure of a relation refers to zero or more applications of that relation connected together.

```
1  // A file system object in the file system
2  abstract sig FSObject { parent: lone Dir }
3
4  // A directory in the file system
5  sig Dir extends FSObject { contents: set FSObject }
6
7  // A file in the file system
8  sig File extends FSObject { }
9
10 // There exists a root
11 one sig Root extends Dir { } { no parent }
```

Figure 3.4: Initial alloy code specifying a basic file system.

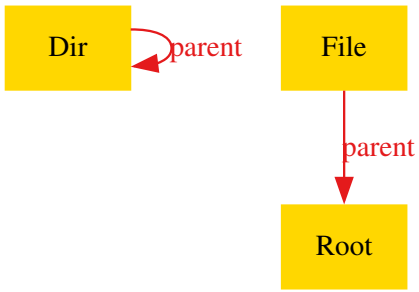


Figure 3.5: An example of a file system that shows a flaw in our initial design.

```
1 // A directory is the parent of its contents
2 fact { all d: Dir, o: d.contents | o.parent = d }
3
4 // File system is connected
5 fact { FSObject in Root.*contents }
```

Figure 3.6: Alloy facts extending the original code to constrain what types of file system can be generated.

```
1 // Property asserting that the contents path is acyclic
2 pred acyclic { no d: Dir | d in d.^contents }
3
4 // Now check it for a scope of 5
5 check { acyclic[] } for 5
```

Figure 3.7: Code showing how properties are defined and checked in Alloy.

Now that we have a model that seems to generate instances as expected, we can use it to check the properties of the design as shown in Fig. 3.7. Using the **pred** keyword we define a predicate on line 2 that specifies that we do not believe that a directory can be nested within its own contents path. In this example, we are using a non-reflective transitive closure operator, ^ ↪ , to only check paths with one or more applications of the contents relation. The **check** command is used on line 4 to ask Alloy to only return instances that violate the listed predicates.

When a check is performed, or any command that prompts the Analyzer to search for instances, a scope is required. Alloy operates on first-order logic, which is undecidable. This undecidability implies that, for an unbounded scope, there is no general algorithm that can determine whether a given property holds across all possible models. Without a bound on the scope, the search space grows indefinitely, leading to infinitely many possible combinations of elements that Alloy would need to check. Setting a finite scope ensures that the analysis remains computationally feasible by manually defining the bounds of the search space.

There are two outcomes to executing a check. If Alloy returns an example, it demonstrates that the behaviour specified in our predicate has occurred within our defined model. If Alloy returns no example, we can use this as evidence that our property holds for the defined scope. Since Alloy's analysis is bounded by this scope, it could always be true that the property does not hold at a larger scope size. While there is no generally decidable means of proving the presence or absence of this, it is observed through the "small scope hypothesis" that, in practice, counter-examples to properties are often found at relatively small scope sizes [115]. In cases where proof is required for all scope sizes, alternative approaches such as theorem proving can be used to complement modelling in Alloy [115].

3.5 Summary

In this chapter, we have given an overview of the concepts and techniques used in the application of formal methods within this thesis. We discussed the background of formal methods and the motivation for their application. We gave an overview of their strengths and provided insight into the challenges in applying them, which has led to the creation of many different approaches targeted at different domains.

We reviewed model checking and its usage in creating mathematical models of systems to

evaluate their safety, reliability and security. We describe how these models are checked using properties, often written in a temporal logic, that describe conditions that the model checker should identify. We then reviewed some common model checking tools and their applications, including UPPAAL, PRISM, NuSMV, SPIN and Alloy. Having considered their key characteristics, we identified SPIN and Alloy as the two approaches to use in our work modelling an ICS system and its DT network.

We concluded by giving an overview of the application of our two chosen modelling techniques. We demonstrated how concurrent processes can be specified in Promela, which can then be verified using LTL properties in SPIN. An introduction to the Alloy specification language is also given, along with an example that shows how instances of models can be generated and analysed using the Alloy Analyzer.

Chapter 4

Case Study: Hydroelectric Dam

4.1 Hydroelectric Testbed

In order for us to be able to develop formal method approaches to a realistic ICS, we constructed a DT framework for a hydroelectric dam testbed. The dam forms part of the University of Glasgow GREENs Testbed.

The dam realistically represents a pumped-storage hydroelectric dam [186]. Industrial pumped-storage hydroelectric dams operate as large-scale electric batteries using water to store electricity. Like all batteries, they are not 100% energy efficient, so some stored energy is lost. However, they are economically viable due to the fluctuating nature of electric prices. Energy is stored as potential energy from the water contained in an upper reservoir or tank. When electricity demand is high, water is released from this upper reservoir by the penstock that transports the water to the turbines that generate electricity before being collected in a lower reservoir. The electricity is sold back to the grid, and the water remains in the lower reservoir until electricity demand drops; low-cost electricity can then power the return feed that brings the water back up to the upper tank. The dam in the GREENs hydroelectric testbed is shown in Fig. 4.1. It is a scaled-down representative test environment containing the OT devices and networks used in industrial pumped-storage hydroelectric dams.

The testbed dam's design before the integration of the DT framework is shown in Fig. 4.2. Two PLCs control the operation of the dam. The generator PLC controls the normal water movement from the upper tank to the lower tank. Within the penstock, the generator PLC controls the upper tank release valve and receives data from the flow rate sensor within the penstock. In the turbine enclosure, the generator PLC controls the pump that pushes the water through the turbine (a requirement due to the small scale of the testbed) while receiving the turbine's voltage output and monitoring the temperature within the turbine enclosure. If the temperature in the enclosure increases too much, the PLC activates the cooling fan to reduce it. The Control PLC controls the movement of water from the lower tank back up to the upper tank. When water is to be returned the value on the lower tank is released and the two sump

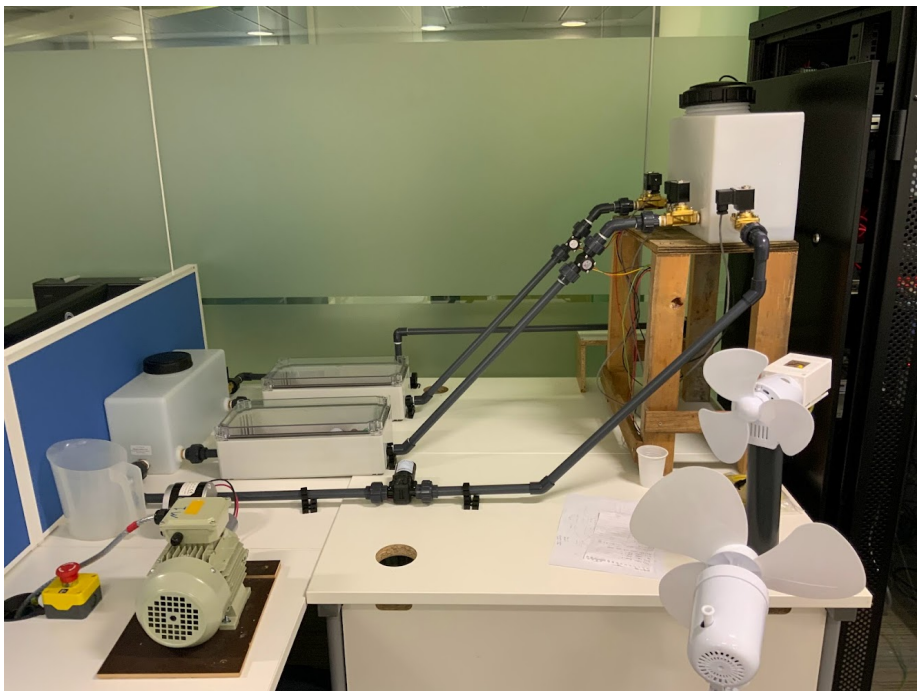


Figure 4.1: Photo of the University of Glasgow GREENs hydroelectric dam testbed.

pumps are activated to push the water up to the upper tank. Additionally, should the water level in the upper tank become too high, the Control PLC can release an additional outflow valve in the upper tank to allow water to be returned to the lower tank through the flood pipe.

4.2 Historian

Communicating with the PLCs in the dam requires the construction of a historian that pulls the data from the memory of the PLCs. Historians are a common supervisory component within ICSs. They gather and collate data from the systems in operation so that operators can monitor device performance and IT devices, usually engineering terminals or corporate enterprise systems, can access it. In our case, a data historian was constructed to contain data that the DT could query; however, in a production environment, the historian may be a pre-existing part of the normal operation of the ICS that is used in several different workflows.

The design of the historian uses three components: a Node-RED [176] flow, an InfluxDB [111] time-series database, and a Grafana [86] dashboard to visualise data. Together they pull the data from the PLCs, collect it and present it to the user. The database the historian is used as the basis for the implementation of the DT elements discussed in this chapter.

4.2.1 Node-RED

Node-RED is a programming tool centered around the construction of data flows. Different APIs and libraries can be combined through a browser-based user interface to allow data to be

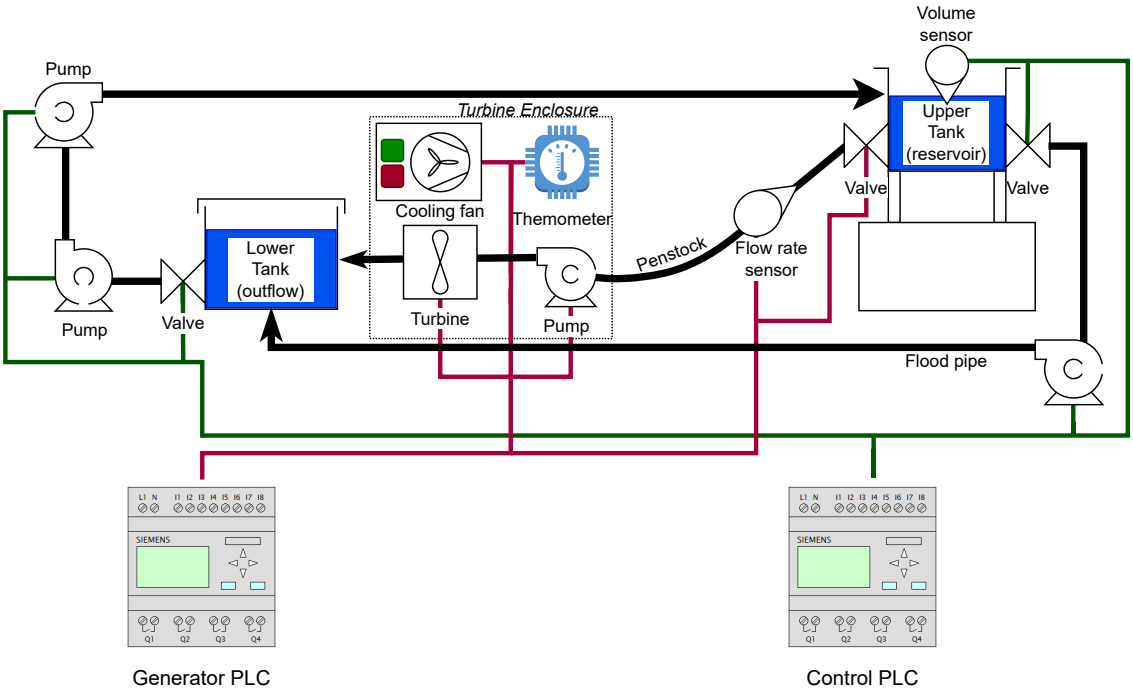


Figure 4.2: System design of University of Glasgow hydroelectric dam.

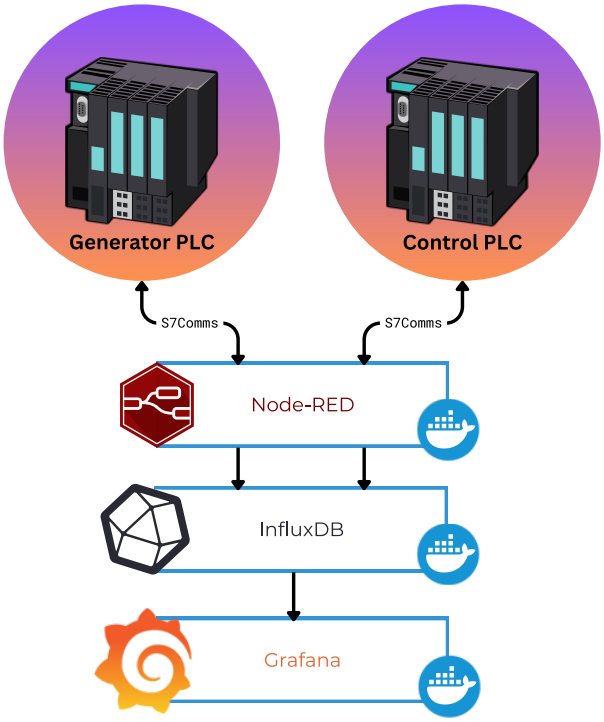


Figure 4.3: The architecture and data flow within the historian constructed for the dam.

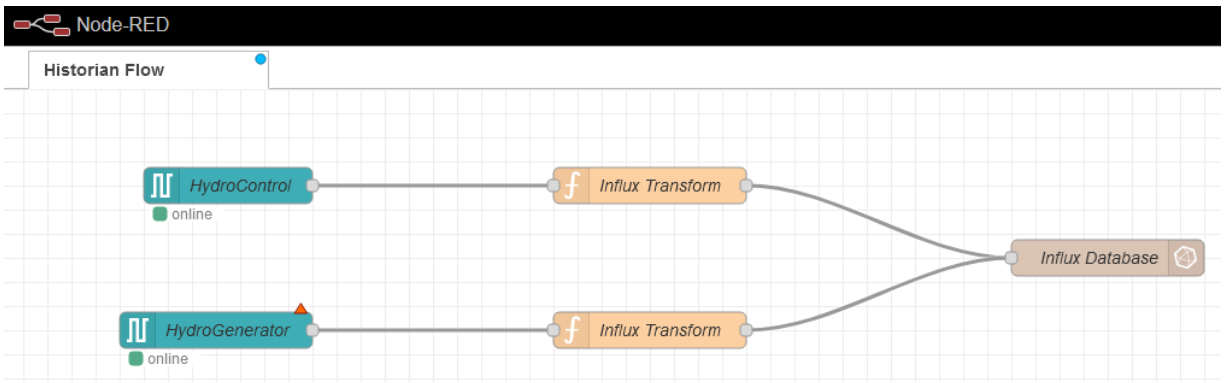


Figure 4.4: Flow of data through NodeRED. Blocks on the left use S7Comms GET requests to pull data from PLCs. The data from each device is transformed into a JavaScript object that is pushed to InfluxDB.

gathered from and moved between locations. The flow used for the historian is shown in Fig. 4.4; it follows many of the steps described in [133] as a foundation with design modifications to gather and store data from multiple sources and adapt it to operate on PLCs from a different hardware provider.

The blue nodes on the left each connect to a PLC in the hydroelectric testbed, querying for a list of tags defined by their name and memory location within each PLC's memory. This is done through the use of the S7 Communication [6] protocol. This data is originally collected as a JSON object so the "Influx Transform" function nodes rearrange this data into a JavaScript object as shown in Fig. 4.5. The constructed objects are then passed to the "Influx Database" node, which sends them to the InfluxDB server. The whole flow is executed for each PLC approximately once every 100ms.

4.2.2 InfluxDB

InfluxDB is a time series database where the values of variables over time can be stored. Data organisation in InfluxDB is performed using buckets. All of the data from the dam's PLCs are stored in a bucket. Within a bucket, measurements can be used to create logical groupings within it. Once data is collected from the controllers, as part of the transform, NodeRED turns the data into a measurement titled with the name of the device that it was collected from. When this data is added to the bucket it is stored together with all the other measurements from that device in the order in which they were collected. Fields within this measurement then contain the values contained within the PLC memory.

4.2.3 Grafana

In the historian, the data in InfluxDB is visualised through a Grafana Dashboard. The dashboard consists of a number of different display modules. All of them query the database for entries

```
// Defining the object that will be sent to InfluxDB
let dbEntry = {};
dbEntry.measurement = deviceName;
dbEntry.fields = {}

// Parse the original message
var plcTags = Object.keys(originalMsg);
// Add tags to database entry
plcTags.forEach((tag) => {
    // Convert booleans to integer representation.
    if (originalMsg[tag] == true) {
        dbEntry.fields[tag] = 1;
    } else if (originalMsg[tag] == false) {
        dbEntry.fields[tag] = 0;
    } else dbEntry.fields[tag] = originalMsg[tag];
});
msg.payload = [];
msg.payload.push(dbEntry);
msg.topic = deviceName + " DB Write";
return msg;
```

Figure 4.5: JavaScript code snippet from Influx Transform function node in Node-RED

that occured during a preset query window, and then display the returned results. The dashboard used during the project is shown in Fig. 4.6. This is an open-source implementation representing the type of dashboard that would be found in an industrial control room. Dashboards are the operator's primary means of gaining an overview of the system's performance. Even from a scaled down representation, it can be seen how the amount of data presented to operators can be overwhelming. For this reason additional monitoring tools are often used to help track the data.

4.3 Digital Twin Framework

In this section we discuss the structure of the DT code and how it gathers data from the historian to present a picture of the dam's state to the operators, and the models that will be discussed later. The system is written in Python 3. After completion of the initial setup and establishing a connection to the database, the system follows a main operation loop. During each loop the system pulls new data from the PLCs, synchronises them into one comprehensive state-space and then queues them to be analyzed by the models.

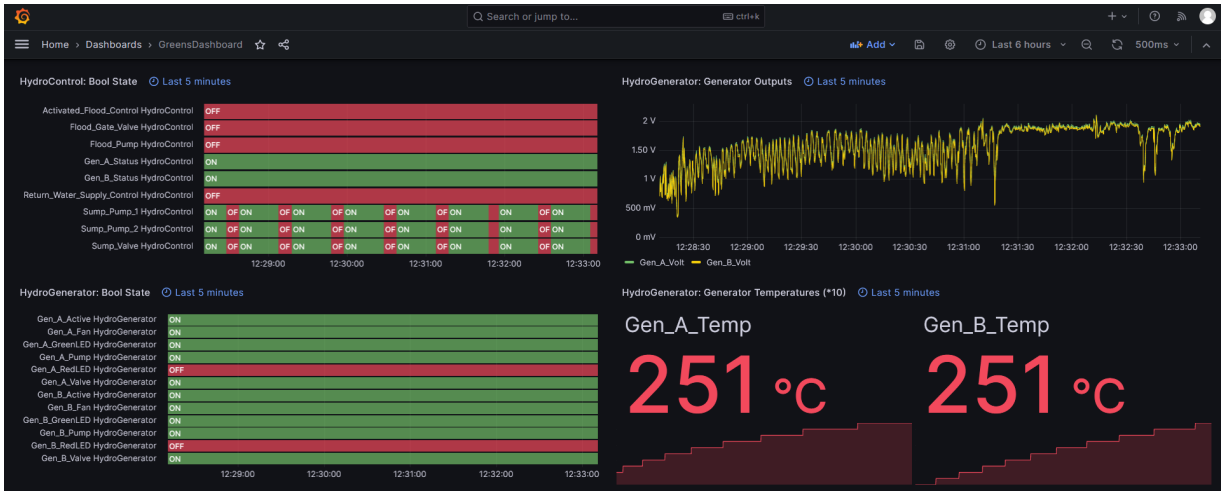


Figure 4.6: The Grafana dashboard displaying generator temperatures and output voltages alongside the binary state variables. Temperatures are scaled by a factor of ten.

4.3.1 Synchronising State Data

The historian provides a means of accessing time-series data about the dam's constantly changing state through the tags associated with different sensors and actuators. These tag-value pairs are updated with new values between 5 and 10 times a second, presenting an interesting challenge when assembling a complete picture of the dam's state. To address this, a python environment was created that could recursively pull this data from the historian, process it and use it to trigger digital models. We will refer to this as the synchronisation module.

With the dam active, the digital twin is triggered by starting this synchronisation module. While in operation, the module queries the InfluxDB database, pulling all the data that was received during the last second. Since the data in the Influx database is separated according to device, this results in two sets of state data; a Control PLC set and a generator PLC set. The task is then to combine these two sets so that a complete state of the dam can be made. This is done through a two-step process.

Firstly we merge rows from the two sets which have the closest timestamps. This creates a single combined dataset encompassing all of the tag-value pairs within the dam. The combined dataset also contains the timestamps of the original data. Secondly, the original timestamps are used to sort the data by preference. In this use case, due to the intention to construct formal models of the dam, a preference was given to sort by how closely aligned the timestamps of the datasets were, modified by how recent the data was. The intention was to ensure that the devices would most agree upon the global state of the dam while still operating on recently gathered data, as even the oldest entries in either dataset would be at most a second old. The scoring method used is shown in Fig. 4.7.

```
def score(record):
    sync_score = abs(
        record['_time_control'] - record['_time_generator']
    ).total_seconds()

    freshness_score = abs(
        datetime.now(tz=pytz.timezone('Europe/London')) -
          min(  record['_time_control'],
                record['_time_generator']
          )
    ).total_seconds()

    return (2 * sync_score) + freshness_score
```

Figure 4.7: Python code taken from the synchronisation function that determines which data readings to combine to provide as model input. The result with the lowest score is used as model input.

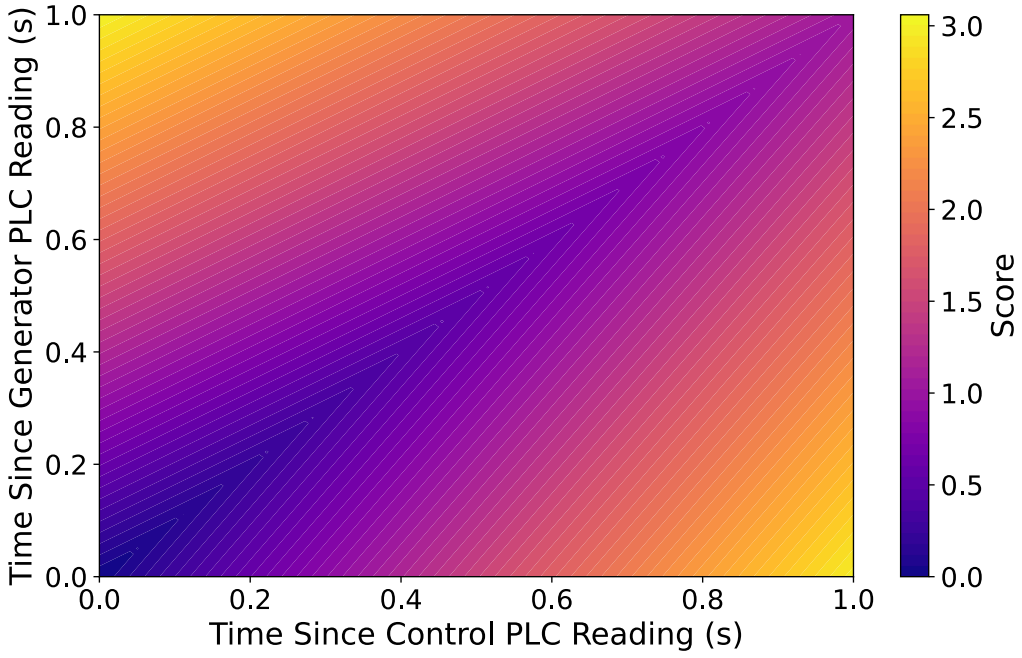


Figure 4.8: A graph showing how the scoring method sorts combinations of results from the PLCs prioritising readings that were written to the database close together, while still including how recently the readings were taken. Low scores are better.

4.4 Developing Anomaly Detection Capability

Our initial goal was to develop an understanding of the normal operational behaviour of the hydroelectric dam through analysis of recorded data from its operation. The DT framework was configured to collect state readings from the PLCs and store them with an index label calculated from the binary representation of the state variables. Each time the system gathered new data from the PLCs the current state of the system was compared to the previous state of the system and a transition between the two states was recorded. Through this approach, state transition diagrams representing the dam's behaviour can be created.

Fig. 4.9 shows an example of a state transition diagram for a subset of the system's behaviours where only Generator A is in operation. From observing the data collected by the dam, it can be observed that despite using a synchronisation system, creating a unified picture of the system can still result in anomalies. In Fig. 4.9, four examples of this can be seen, shown in red. These states contain conflicting information about whether the generator is in operation or not, where Gen_A_Active and Gen_A_Status are not equal.

Collating data from multiple sources, even while making efforts towards synchronisation, presents a challenge in this instance as one set of tags in the Control PLC tracks the value of tags in the generator PLC. Gen_A_Status and Gen_B_Status are Control PLC tags that track the values of Gen_A_Active and Gen_B_Active within the generator PLC's memory. During each cycle, the generator PLC inserts the value of Gen_A_Active and Gen_B_Active into the Control PLC's memory locations for Gen_A_Status and Gen_B_Status. However, the Historian and, subsequently, the DT can receive conflicting information about the state of the Generator. When a change occurs in the state of one of the generators, this update process between the PLCs can be interwoven by NodeRed's GET request to pull the variables. The result is that the updated value of, for example, Gen_A_Active, is received alongside the old value of Gen_A_Status. A message sequence chart demonstrating this behaviour is shown in Fig. 4.10. A model based entirely on the PLC code, as written, would not recognise that this behaviour can occur, and it would, therefore, be flagged as an anomaly. However, since this is known to be a symptom of the normal operation of the historian infrastructure, any anomaly detection system needs to represent both the logic of the PLC code and the infrastructure used to connect it to the PLCs. In doing so, the system will be able to accept these desynchronised states, allowing it to be a more useful indicator of legitimately anomalous behaviour.

The infrastructure already developed can be used as the basis of an anomaly detection system. The observed states and transitions from our testing can be used to establish a system baseline, i.e., a set of states that reflect normal behaviour. In a simple example, as observed states are identified, they can be compared to this baseline of previously observed behaviour. If the state has been observed before, it can be ruled out as an anomaly. The drawback of this method is that it depends upon being able to gather a complete set of system baseline data from the system without any risks of the system being tampered with during that process. Since

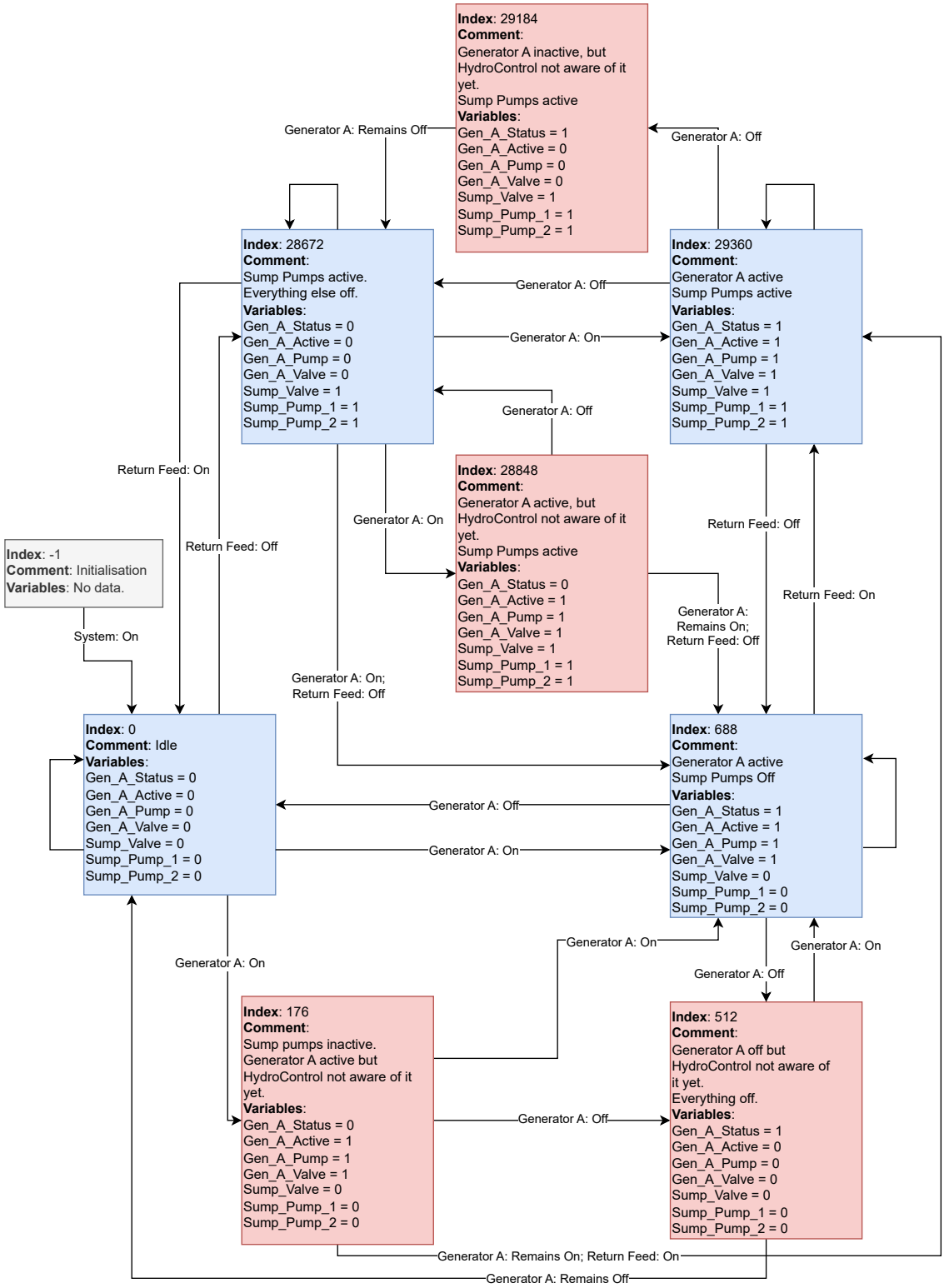


Figure 4.9: A state transition diagram representing the behaviours of the Dam when only Generator A is active. Blue states show consistency between Gen_A_Active and Gen_A_Status, Red states do not.

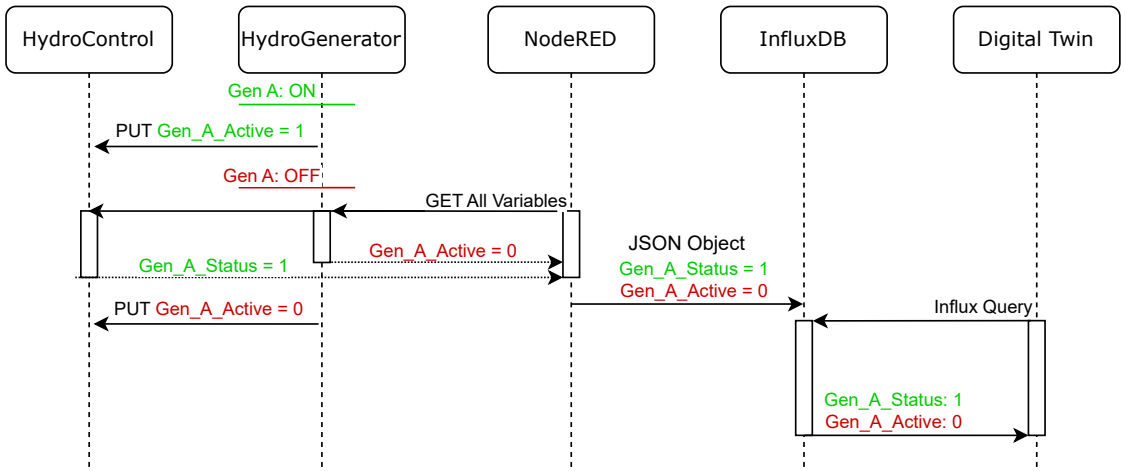


Figure 4.10: A message sequence chart showing how the PUT requests from the generator PLC that synchronise Gen_A_Active with Gen_A_Status can result in the DT receiving conflicting data about the state of the dam.

the approach relies on observational data, rather than inspectable artefacts, it is susceptible to supply chain attacks that modify the behaviour of the system before the point of deployment. Additionally, the collection of a comprehensive baseline of every reachable system state could rapidly become infeasible in a large-scale industrial system. Even for us, testing the approach on a testbed environment with a reduced scale and collecting a complete baseline of the system in every possible configuration proved challenging. We, therefore, conclude that a more sophisticated method of determining the validity of observed system states is required. Since similar observation-based methods, such as training a machine learning model, would also depend primarily upon this system baseline, we propose an approach based on the executable code programmed within the controllers.

4.5 Summary

In this chapter, we constructed a DT framework capable of connecting digital models to a hydroelectric dam testbed. We present an introduction to the GREENs Testbed hydroelectric dam, describing its purpose, components and architecture.

We explain the process of constructing a historian, which consists of three components: NodeRED, InfluxDB, and Grafana. NodeRED was used to interface with the PLCs, pulling the data through S7Comms requests before transforming it into JSON objects to be inserted into the database. The time-series database InfluxDB receives data from NodeRED and stores it to be accessed by Grafana, which then visualises the state of the dam.

An additional DT module was integrated into the ICS environment to retrieve data from the InfluxDB database and synchronise state data collected from two PLCs. We explain how a scor-

ing metric determined the optimal combination of PLC states based on their generation time and timestamp alignments. This module constructs a unified representation of the dam's operational state, making it available to models within the framework. We established a baseline of known dam behaviours by recording this state data. Tracking this combined state data then guided the search for a practical approach to representing and identifying normal system behaviour.

Chapter 5

Specification-Based Anomaly Detection In SPIN

In this chapter, we use Promela to create a state transition system for the dam. Promela was chosen because it is designed to model concurrent processes and analyze their interactions. Our system consists of multiple co-executing devices, including two PLCs that operate concurrently to maintain the system state and an HMI through which the operator provides commands to the system.

Unlike that presented in chapter 4.1 (Fig. 4.9) our Promela model represents a full specification of all of the system components, directly based on the logic of the executable code written to the PLCs. Using SPIN, the actions of all the components can be combined to define the complete space of reachable system states. This allows the model to extrapolate to unseen behaviours, reducing the requirement for comprehensive system baselining before deployment since the only need for system data is to test the system. Additionally, this approach is not limited to detecting only known anomalies. Using SPIN, we only need to specify the types of behaviour the system is programmed to perform, and any deviations from that will be detected by the system.

5.1 Approach

Our method of connecting the historian data to the SPIN model checker is shown in Fig. 5.1. We represent the system's behaviours in a Promela template containing placeholder values for each tag value in the state data. Each time the DT framework collects and assembles the most recent snapshot of the dam's state, it embeds it in the template to drive the analysis. The property we use specifies that the currently observed state of the dam is not reachable. SPIN will then analyse the file to determine if a counter-example allows the dam to enter that state. If one is found, the state is recognised by the model, providing evidence that the system is exhibiting normal operational behaviour.

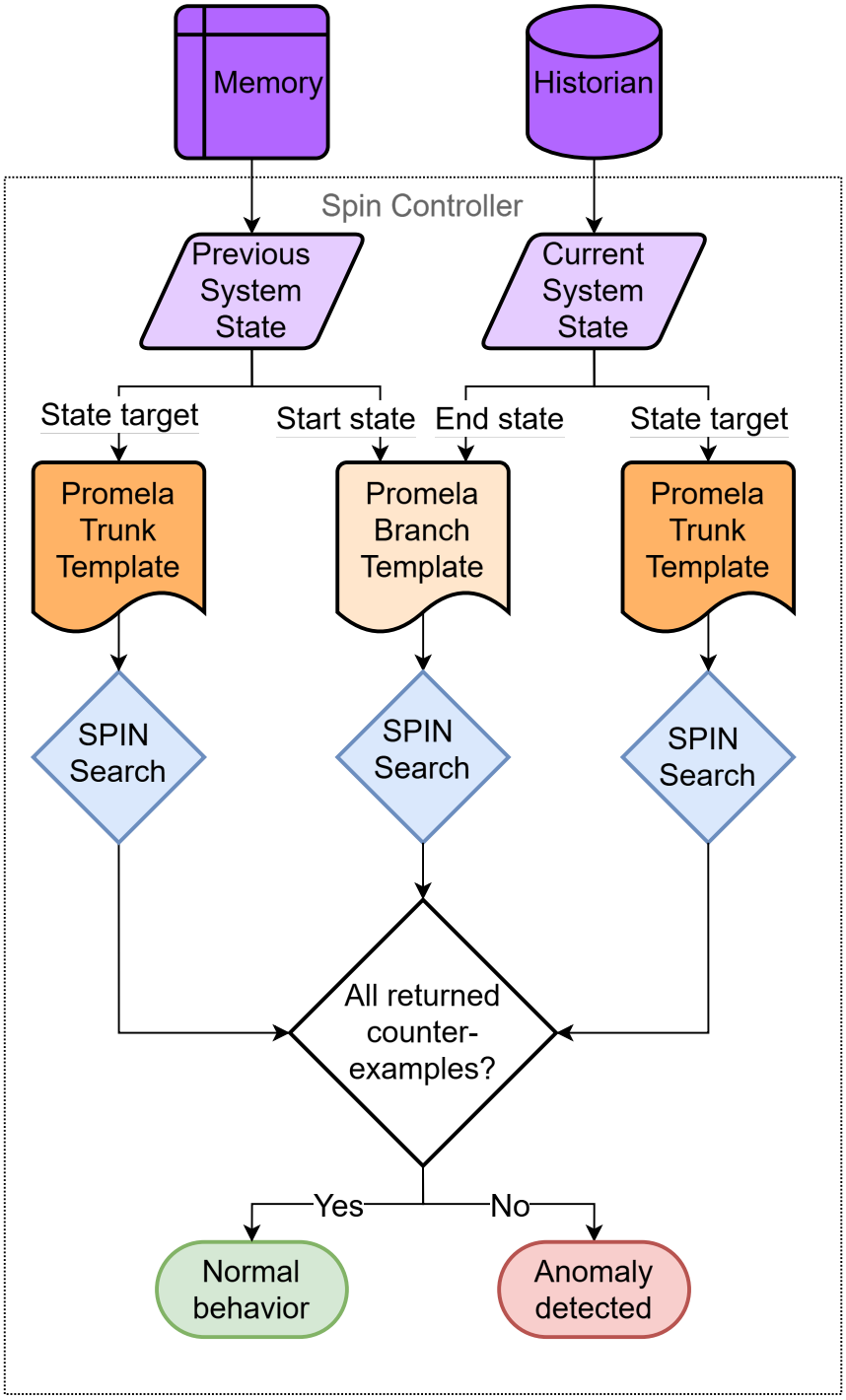


Figure 5.1: The proposed approach for detecting anomalous behaviour in the Hydroelectric dam using SPIN. Two templates are created, one each for the trunk and branch models, containing placeholder values for system state data. The SPIN controller module embeds the state values into these files before executing a search for counter-examples in each. If all of the models return counter-examples the observed system states and transition between them is considered valid. If any model cannot find a counter-example it indicates an anomaly.

Our method uses two models to reduce the state-space and decrease the time taken to identify anomalies, thereby enabling the detection of anomalies in real-time data. The two models target different aspects of anomaly detection and are used to present a complete analysis of the system's behaviours. To explain our method, consider the dam as a state transition system where each state represents a valuation of the state variables. The system moves through the different states by executing transitions. When the system starts, it begins in the same state each time, but as it runs, the number of possible states and traces to reach those states grows larger, branching out like a tree to create an exponentially increasing number of execution traces. To detect anomalies, we must consider whether the state we have observed is a valid system configuration and whether it is consistent with the previously observed system behaviour. Identifying if a state is valid in isolation can be achieved with a relatively simple property, stating that the observed state never occurs and prompting SPIN to show a trace where it does. However, the second part, evaluating the reachability of a second state after a first state has been reached, is more challenging, requiring the traversal of each path that can lead to that initial state and then each branch from that state. By dividing these two aspects of anomaly detection, we can reduce the size of the problem being analysed and focus the models on the specific element of the system behaviour being checked. Our first model determines whether the observed state is reachable by any valid execution trace. We refer to it as the "trunk" model since it forms the basis for our analysis. Our second model is designed to identify anomalous transitions between states. Taking the state checked by the trunk model as a starting point, we use a second model to determine if any branches originating from it can reach the next observed state. Finally, we also check if the trunk model recognises this next observed state. If a SPIN search of any of these models cannot find a counter-example, then the system is behaving anomalously because one of the states is irregular, or the transition between the two states should not be able to happen.

5.2 Modelling PLC Code in Promela

In this section we discuss the development process of creating a Promela representation of the ladder logic programs in the PLCs. The complete PLC code and its Promela representations are too large to be wholly included in this thesis. To demonstrate how the system was modelled, we instead show how the structure of our Promela model captures the PLC programs' logic and examine a couple of key excerpts from the PLC programs to use as examples of our transcription process.

5.2.1 Ladder Logic

The PLC code is written in ladder logic, a graphical programming language derived from the diagrams used to represent the combinational logic of relay racks. A program is anchored to a vertical "Rail" that runs down the side of the program onto which "rungs" can be attached. To

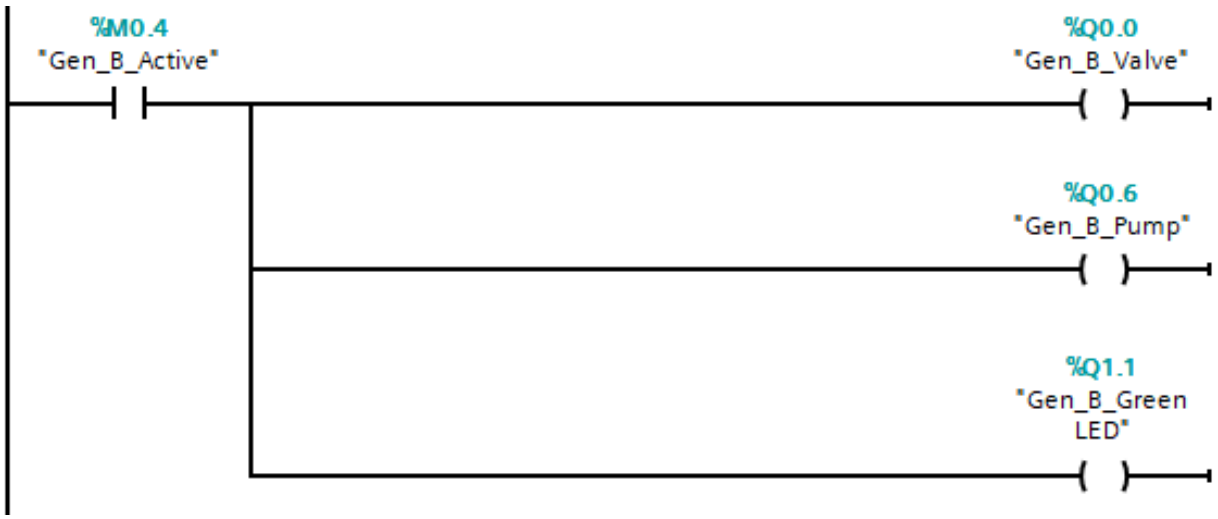


Figure 5.2: An example of a ladder logic rung. When Gen_B_Active is ON, each of the three outputs is set to ON. When Gen_B_Active is OFF, each output is set to OFF.

understand this code, a rung can be considered as a line of code; it defines a control sequence of operations. While the code aims to replicate the electrical schemas of relay racks, the rungs do not behave like electrical circuits. Instead, the PLC rapidly executes these rungs during each scan cycle, replicating the effect of simultaneous execution.

Fig. 5.2 shows a simple rung from the generator PLC code that controls the actuators within the generator turbine enclosure. The program responds to changes in Gen_A_Active and Gen_B_Active as the operator changes them through the HMI (not shown). When either variable is set to true, the rung in Fig. 5.2 opens the corresponding valve to release water from the upper tank, turns on the pump to push the water through the generator turbine, and turns on the green LED to signal that the generator is on.

5.2.2 Simultaneous Execution

Translating PLC code to Promela comes with the inherent challenge of capturing the simultaneous execution of the PLC instructions within the bounds of a sequential Promela program. SPIN will execute the code in sequential order when searching for matching states, meaning that all updates to variables within each rung must be simultaneously affected. This simultaneous assignment to these variables can be done using a `d_step` operation. This instructs SPIN to execute all of the commands within the model in a single instruction. Therefore, when the state-space is searched, no state will exist for the intermediate assignments within the state-space. Our design approach, therefore, is to perform the logical computations of the rung and then write the outcome as a single `d_step` operation. As an illustration of this approach, we refer to Fig. 5.3, a simplified excerpt from our hydro_control process in Promela that represents the ladder logic shown in Fig. 5.2. The model identifies which of the rungs are active and inactive in the PLC through the use of a conditional if statement. The output of this is a deterministic selection of

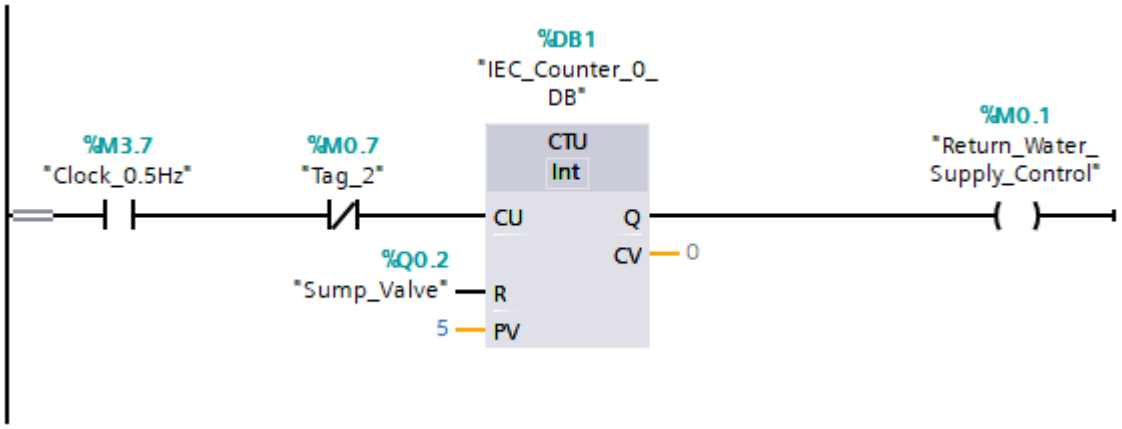
```
1  if
2     /* Identify the combination of rungs to be executed */
3     :: Gen_B_Active == true -> goto Generator_B_Control
4  fi;
5  ...
6  /* Execute outcome */
7  Generator_B_Control: skip
8  d_step {
9     Gen_B_Valve = true;
10    Gen_B_Pump = true;
11    Gen_B_GreenLED = true;
12    Gen_B_RedLED = false;
13 }
```

Figure 5.3: Excerpt from the control PLC model in Promela, simplified to highlight how the rung in Fig. 5.2 is represented. The initial **if** statement identifies the state the system is currently in and then jumps to the **d_step** code section, where the relevant assignments of variables are performed.

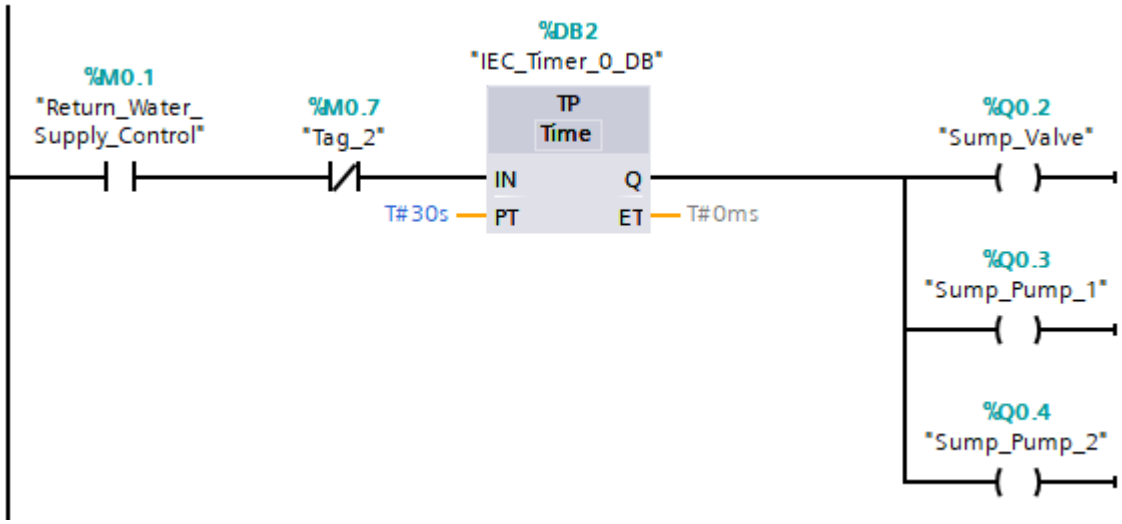
the appropriate **goto** statement that transfers the control to the relevant section of the process where the outcomes can be executed. In this case, the label Generator_B_Control is used. SPIN doesn't allow control to jump into a d_step sequence, so using a **skip** statement is required. Inside the d_step statement, an assignment to each of the required variables is performed. This includes operations that are affected by Gen_B_Active being false, such as turning the red LED off. If the d_step instruction was not used, valid states could be found in the intermediate states. For example, if the generator had previously been off and was just turned on, SPIN would incorrectly recognise states where Gen_B_Active and Gen_B_Valve were true, but Gen_B_Pump and Gen_B_GreenLED were false (since they are updated after Gen_B_Valve). The effect of this only increases in line with the number of variables to be updated in the system.

5.2.3 State-space Reduction

While the method requires the accurate representation of the PLC code, steps must be taken to reduce the size of the state-space. In Fig. 5.4, a ladder logic snippet from the control PLC that controls when the water return feed activates to bring water from the lower tanks back up to the upper tank is shown. The return feed consists of two sump pumps (labelled Sump_Pump_1 and Sump_Pump_2) and a normally-closed valve (Sump_Valve) that seals the water return pipes from the upper tank while the return feed is inactive. The single return feed is used differently depending on whether one or both generators are active. To balance the water usage of a single continually running generator, the return feed remains off for 10 seconds before activating for 30 seconds and then repeating. If both generators are in use, the system only waits 5 seconds before executing. The wait and active intervals are measured, respectively, using a counter that



(a) Rung that counts five seconds before triggering Return_Water_Supply_Control.



(b) Once Return_Water_Supply_Control is triggered, a 30-second timer is started, and the actuators of the water return feed are activated. Once the timer elapses, the return feed is stopped.

Figure 5.4: Control PLC Ladder Logic rungs that work together, alternately waiting 5 seconds before triggering the return feed for 30 seconds. Tag_2 is a testbed feature used to trigger anomalous behaviour, and it is normally OFF.

counts the number of times a 500 ms clock activates and a timer that counts 30 seconds from when the feed is activated.

A key objective when constructing our representation of the PLC is to reduce the number of operations required to execute the same piece of control logic. While a PLC can execute the rungs of its ladder logic code multiple times a second, our SPIN representation cannot afford to perform the same number of actions. It is, therefore, essential to focus on the key behavioural changes that the code is performing. A big step towards this is using a restricted representation of timers to reduce the state-space. Instead of executing code to increase timers by 1 step each loop incrementally, our representation increased them by 10 each time, dramatically reducing the size of the search space required to determine if the behaviour is valid while still including the control logic used in the operation of the timers.

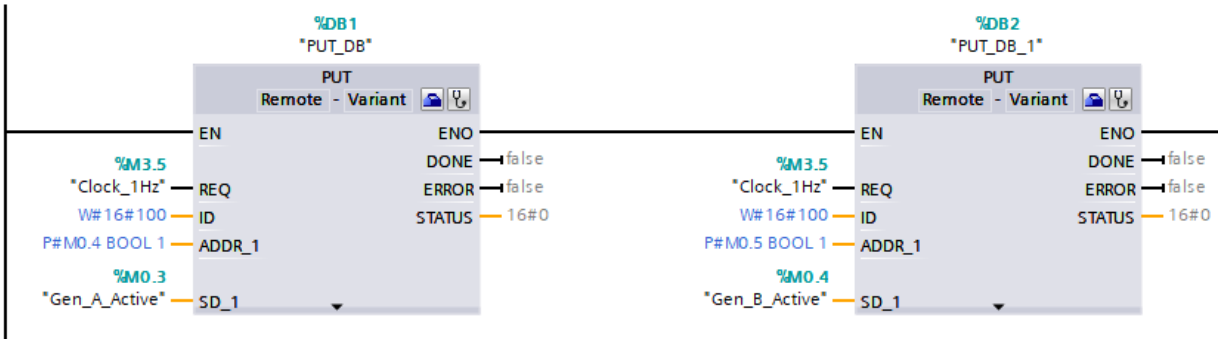


Figure 5.5: Excerpt from the generator PLC showing the PUT functions used to update the control PLC on the status of the generators.

5.2.4 Device Communication

Promela supports interprocess message passing through the use of channels. These are defined with a specified capacity, denoting the number of messages that a channel can contain. Data is retrieved from channels on a first-in-first-out basis. Processes that try to retrieve data from an empty channel will block until a message is added to it. Conversely, processes attempting to add messages to a full channel will block until a message is removed. While this was considered as a means of modelling the communication between devices this blocking behaviour is inconsistent with the behaviour of the PLCs. The PLCs exchange data through the use of PUT and GET instructions within the S7 Communication protocol; these instructions insert the value of the data into the memory of the receiver at the specified location without the receiver needing to execute any logic to receive or process it. Therefore, identifying a suitable means of integrating, removing, and processing status messages between the PLCs became challenging and involved considerable deviation from the written PLC code. Instead, we used global variables that allow the processes to modify each other's variables directly. This method means that messages are exchanged instantaneously. As SPIN executes Promela commands non-deterministically, the sending of the data may be delayed, which partially represents the transmission delay of message exchanges.

Two different Promela models were created to model the dam, a full state-space representation, and a restricted transition-focussed model.

5.3 Development of Branch Model

To verify the reachability of the changes of state between the system state snapshots that are collected by the DT we derived a new model from the original Promela model. This new model is referred to as the "branch" model in reference to its start state being attached to the same state that was analysed for reachability with the trunk model. This model starts from the same state analyzed for reachability in the trunk model, but it is designed to capture the set of states that

```
1  /* Generator A on, run water control */
2  Water_Return_Single_Gen_A: skip
3  Return_Water_Supply_Control = false;
4
5  /* Turn off all valves and pumps while waiting to activate */
6  d_step {
7      Sump_Valve = false;
8      Sump_Pump_1 = false;
9      Sump_Pump_2 = false;
10 };
11
12 /* Check for exit triggers */
13 if
14 :: Control_PLC_MODE == false -> goto id_ctrl_state
15 :: Gen_A_Status == false -> goto id_ctrl_state
16 :: Gen_B_Status == true -> goto id_ctrl_state
17 :: HMI_Return_Feed == true -> goto id_ctrl_state
18 :: else -> skip
19 fi;
20
21 /* Wait 20 seconds before turning on return feed. */
22 /* Counter does not increase if Tag_2 is true */
23 if
24 /* Counter increase by 10 instead of 1 reduces state-space */
25 :: IEC_Counter_0_DB < 20 & Tag_2 == false ->
26     IEC_Counter_0_DB = IEC_Counter_0_DB + 10
27 :: IEC_Counter_0_DB == 20 & Tag_2 == false ->
28     IEC_Counter_0_DB = IEC_Counter_0_DB + 10 ;
29     Return_Water_Supply_Control = true
30 :: Tag_2 == true -> skip
31 fi;
32
33 /* Loop or activate water return */
34 if
35 :: Return_Water_Supply_Control == false ->
36     goto Water_Return_Single_Gen_A
37 :: else -> skip
38 fi;
```

Figure 5.6: Excerpt from the representation of the control PLC in the trunk model showing how timers are modelled. Time is discretised into intervals of 10 seconds to reduce the state space.

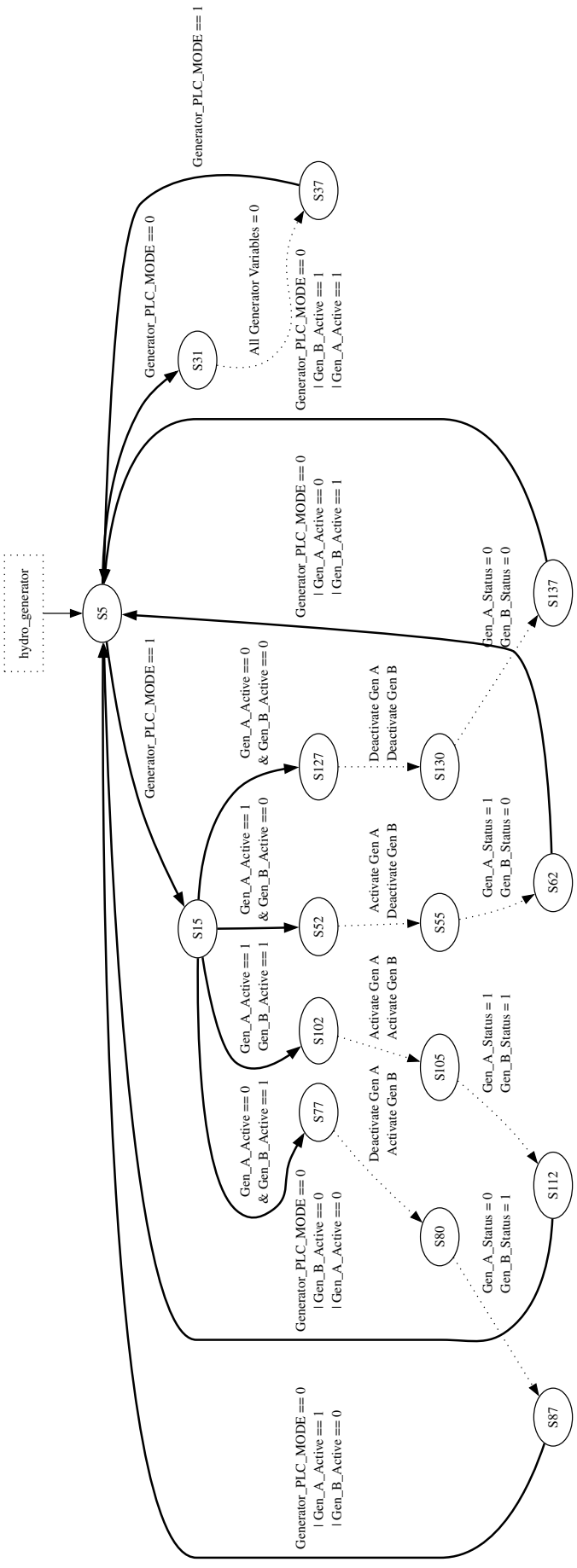


Figure 5.7: A state transition diagram of the generator PLC modelled within the full state-space model.

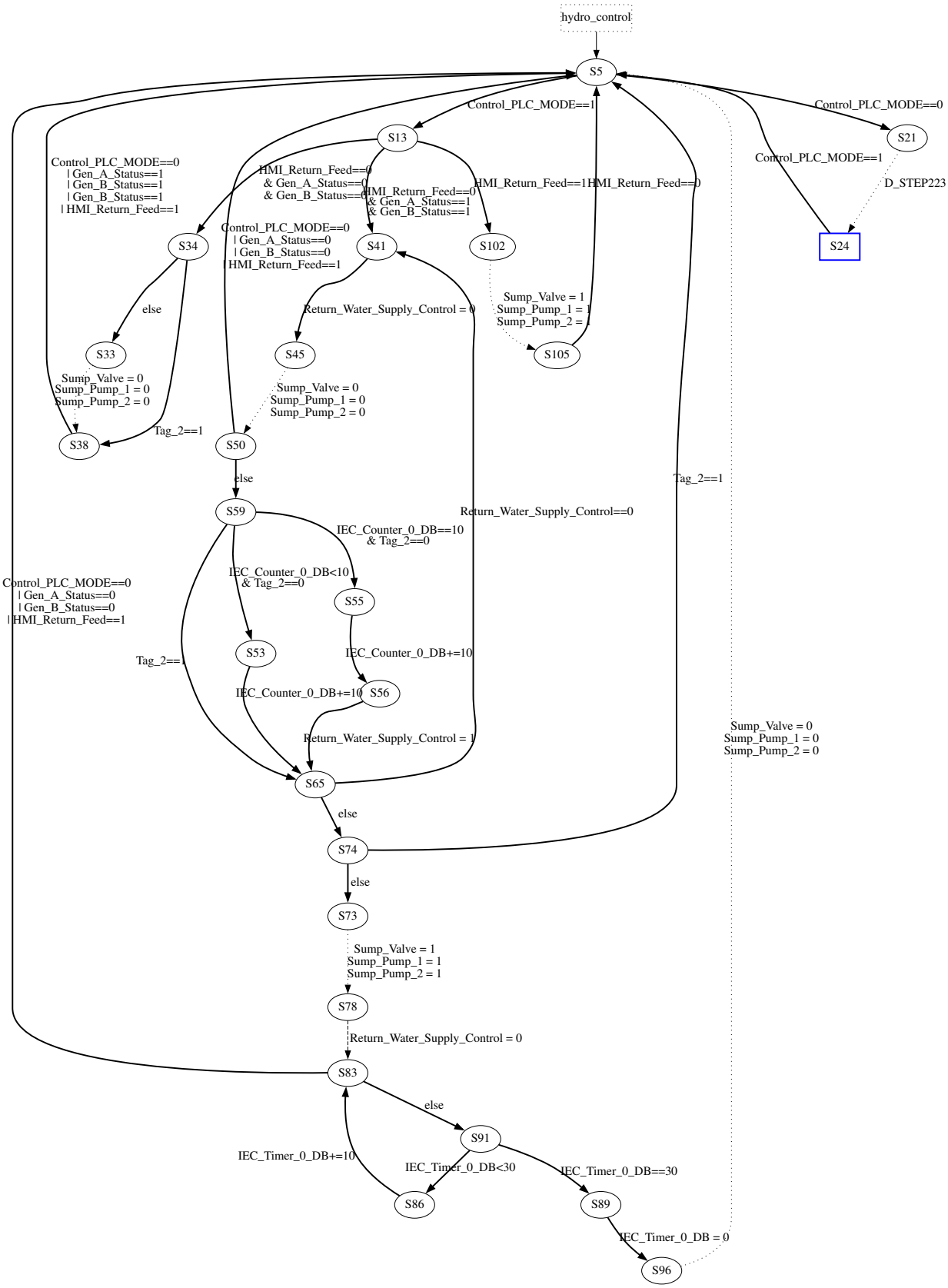


Figure 5.8: A state transition diagram of the generator PLC modelled within the full state-space model.

can be reached from that start state. This approach aims to restrict the state-space to those states that can be reached within a 0.8 s time interval (the time taken to gather new data). We use this approach to verify the model's behaviour in the context of its previously observed state.

This approach's development required considering how to construct a time-restricted variation of the original model. Promela has no provision for modelling time, yet seemingly valid system states may be invalid if they are inconsistent with recently observed system behaviour. An operator might turn a generator off or activate a return. However, they cannot turn both generators off while enabling a manual return of all water within the time it takes for the system to pull new state data. So, while a state with both generators being off and the manual return feed active might be a valid and reachable state for the system, it is anomalous in this context, given how rapidly the system has shifted into that mode.

In the context of a state transition system, this involves reducing the set of states considered to a subset of reachable states. Fig. 5.10 shows an outline of the state transition diagram of the Promela model that represents the generator PLC. Contrasting this with the model shown in Fig. 5.7 shows how the state-space is reduced to focus primarily on the shifts between operational modes. This was done by focusing on key state variables that indictate that a state change is occurring in either of the generators. In the case of the generator PLC these changes are driven by instructions from the HMI, modelled as a separate process.

As discussed previously, the program code in this system involves the use of timers to trigger the transitions, as well as HMI inputs from operators. In the branch mode model, we use fully non-deterministic transitions between these states to represent timers since the historian does not gather timer data. When a time interval is present in the code, we insert an unguarded `if` statement to allow the program to represent both the outcome of a timer completing its time interval and a timer continuing to count. s

5.4 Evaluation

To evaluate our approach, we tested its reliability in identifying anomalous system states from baseline states. This was done by repeatedly running the SPIN models in an isolated test environment on test states and transitions. A full evaluation of our system would determine how our models classify every state that the system could reach and state change that the system could exhibit. A state consists of 23 binary variables yielding 2^{23} $(8,388,574)$ possible state variations and 2^{46} (7.03×10^{13}) possible transitions between those states. Since testing all of these states is infeasible, a more selective approach is required to evaluate how well the model identifies anomalous states.

Our evaluation methodology and its results are presented in Figs. 5.12, 5.13 and 5.16. We constructed a bank of test states that are closest to the baseline states observed during testing since, theoretically, these would be the most subtle anomalies that would be the most challenging

```
1  proctype hmi()
2  {
3     /* Allow PLCs to compute non-operator transitions first */
4     run hydro_generator()
5     run hydro_control()
6
7     /* Perform any operator interaction */
8     send_instruction:
9     atomic {
10       if
11       /* Turn either generator on, if off */
12       :: Gen_A_Active == false -> Gen_A_Active = true
13       :: Gen_B_Active == false -> Gen_B_Active = true
14
15       /* Change the generator PLC into RUN mode */
16       :: Generator_PLC_MODE == false ->
17                                Generator_PLC_MODE = true
18
19       /* Activate manual return on control PLC */
20       :: HMI_Return_Feed == false -> HMI_Return_Feed = true
21
22       /* Operator takes no action */
23       :: skip
24       fi;
25    };
26
27    /* Loops operator to perform at most two actions */
28    if
29    :: OperatorAction == false -> OperatorAction = true;
30          goto send_instruction
31    :: OperatorAction == true -> skip
32    fi;
33 }
```

Figure 5.9: Excerpt from the HMI process representing user interaction. The main "send instruction" loop represents an extensible list of possible ways that the user can interact with the system. It has been reduced to save space; what is shown is a cross-section of the full list used in testing.

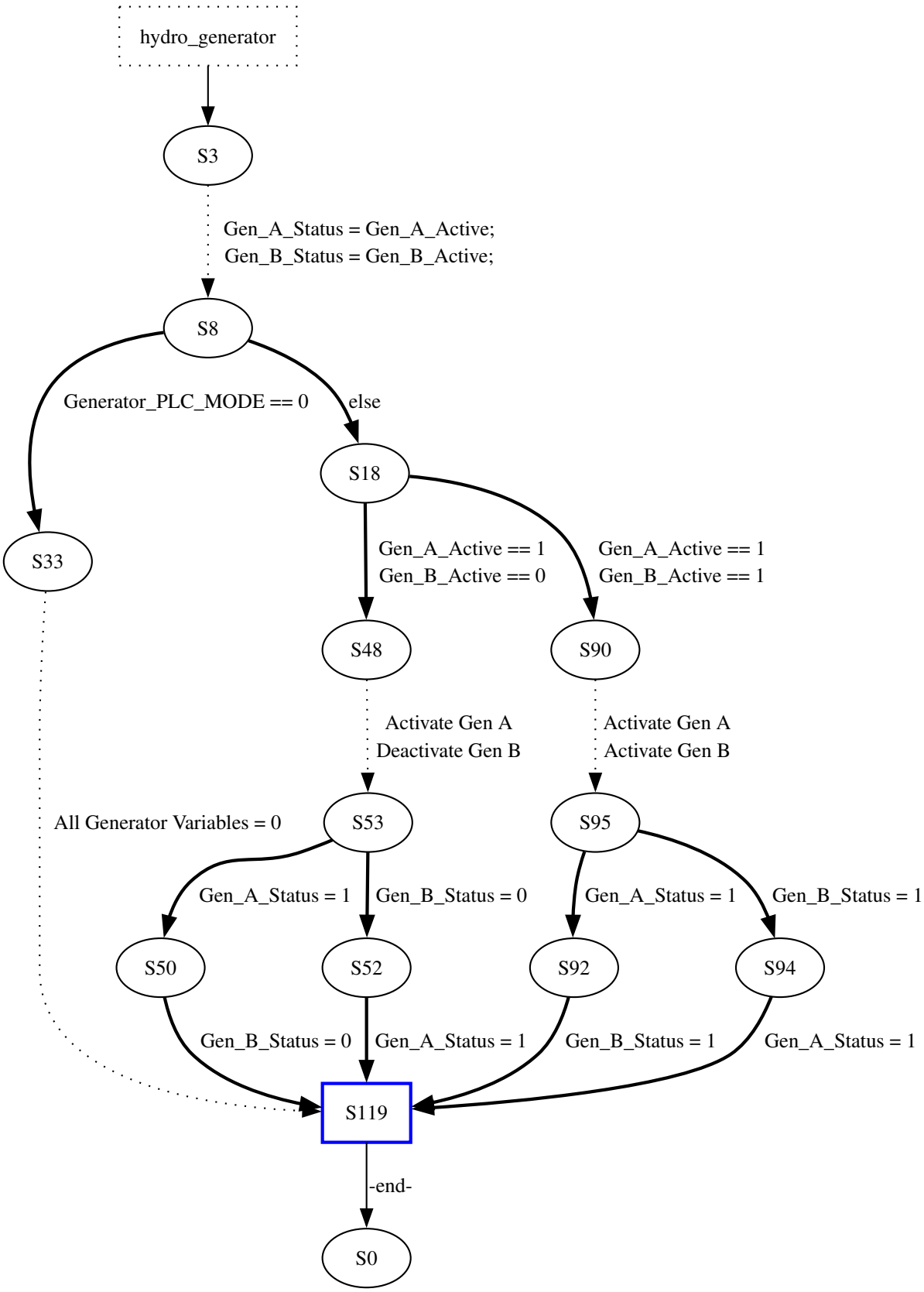


Figure 5.10: A state transition diagram of the generator PLC modelled within the transition model. The model specifies behaviour changes, not the entire state-space. Some extra branches (originating from S18 and connected back at S119) have been omitted for readability.

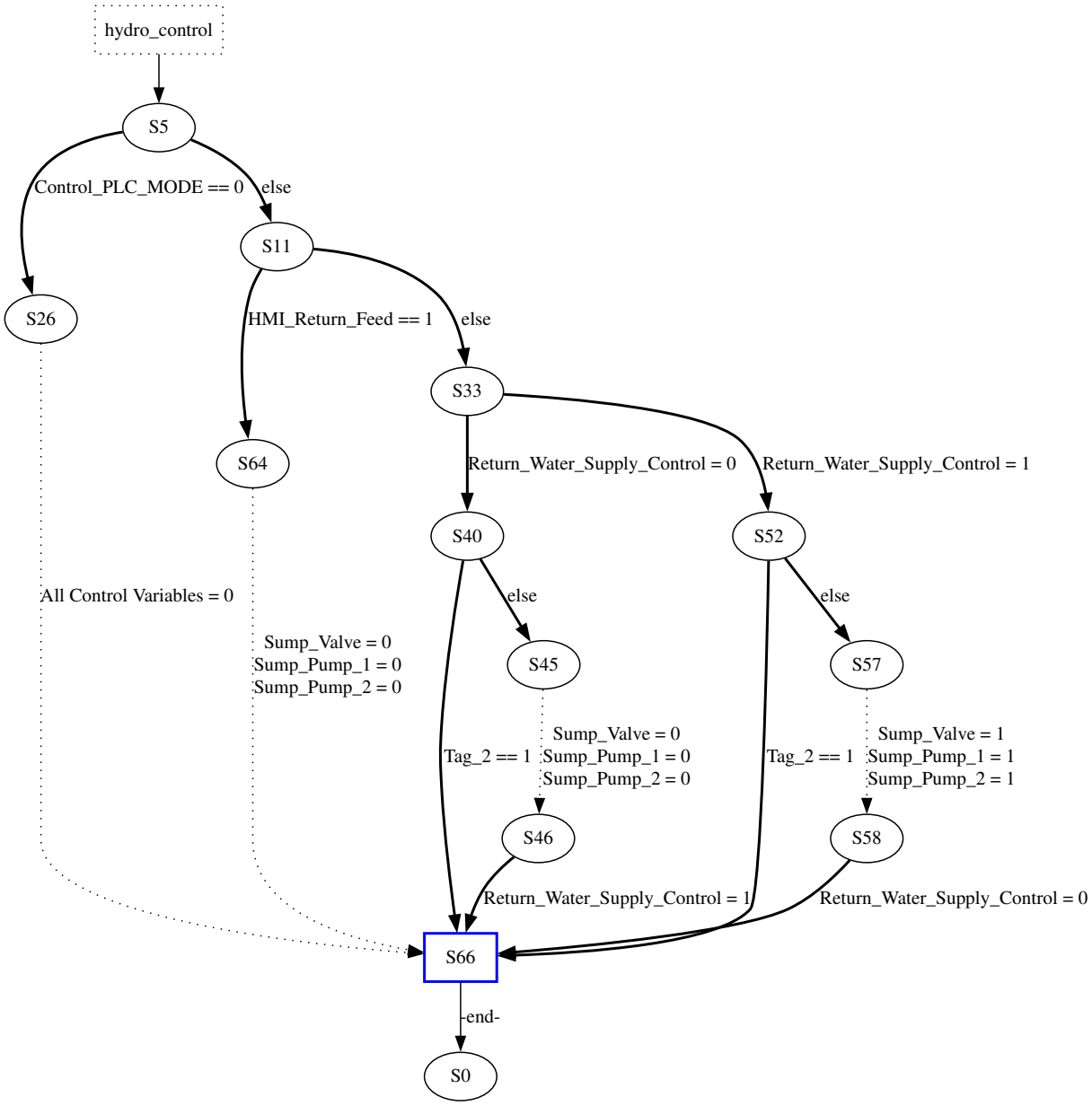


Figure 5.11: A state transition diagram of the control PLC modelled within the transition model. The model specifies behaviour changes, not the entire state-space.

to distinguish from normal behaviours. Our evaluation assumption is that these subtle variations of normal behaviours are where errors in our approach are most likely to be found. If the models can identify deviations in these small changes, it is more likely that bigger changes would also be identified.

5.4.1 Trunk Model

We used the baseline set of states as an initial sample to generate our first set of anomaly states, containing each state that could be made by modifying a single-bit value in a baseline state. For example, the state 0 has every tag value set to 0; if it was present in the set of baseline states, our first set of fabricated states would include every state where a single tag value is set to 1. Duplicate states were removed, and the process was repeated to generate a set of states where two bits were modified, then two more sets for three and four modified bits, respectively. At this point, the exponential increase in size becomes too great to viably evaluate sets with a larger number of deviations (the set of 5-bit deviations has size $359,205$ and would take a month to analyse by itself). Combining these four anomalous sets of states with the baseline states gives a set of $122,216$ test states consisting of the most subtle state changes an attacker could attempt to put the system into. The results of analysing these test states with the trunk model are shown in a Sankey diagram in Fig. 5.12. The inputs to the model are shown on the left side with the output of the model being broken down into their respective groups as they progress to the right, where the results are shown. A flagged state is one where the model did not produce a counterexample demonstrating the reachability of that state, thereby flagging it as an anomaly. We evaluate how well our model performs by how often it is correct. Each output can be categorised into one of four categories. We are using our model to test for anomalies. Therefore, a positive is an input that is identified as an anomaly, and a negative is an input that is considered normal and not flagged as an anomaly. The true and false prefixes denote the true ascribed nature of the input. Therefore, a true positive is a correctly identified anomaly, and a false positive is a normal baseline behaviour incorrectly flagged as an anomaly. Conversely, a true negative is a correctly recognised normal baseline behaviour, and a false negative is an anomalous behaviour that wasn't identified.

While examining the results of the trunk model evaluation, an unused PLC tag was discovered in the set of the test states. This tag, `Activated_Flood_Control`, is declared within the PLC code but never used and does not control anything. It is also not used within the Promela model. However, as it is a variable within the definition of a state, it leads to duplicate states that are effectively equivalent since whether this value is 1 or 0 has no impact on the actual start state of the system. Additionally, these had no impact on the results of the SPIN analysis since the value of this tag was not included in the property being searched for so a state would be recognised or flagged regardless of the value of `Activated_Flood_Control`. For transparency, these states have been identified and removed as "Duplicates".

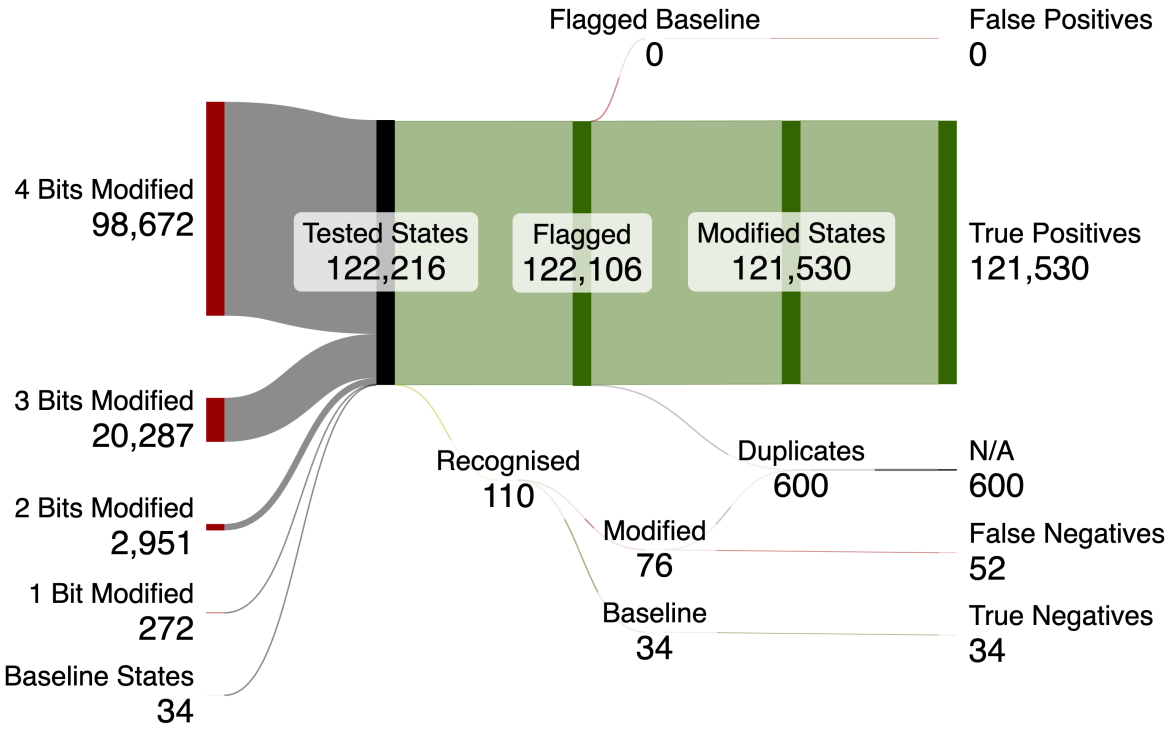


Figure 5.12: A Sankey diagram of the inputs and results of the evaluation of the trunk model. The diagram is read from left to right. The left side shows the make-up of test states. Moving right shows the number of states that were flagged as anomalies and recognised as normal behaviours. These groups are then categorised into true positives, true negatives, false positives and false negatives based on whether the model's output matches their true classification.

While a high detection rate for anomalies is a promising result for the trunk model, the recognition of a small number of fabricated states is a concern. These are not states that have been observed as occurring within the dam and, as a result, could represent potential attack states that the model would fail to warn users about. There are two possibilities: either these are attack states that our model is failing to recognise, or they are legitimately reachable states that our testing hasn't uncovered. To address this issue we first identified these states and then tested them in our evaluation to determine whether they would be recognised as being valid by each model.

5.4.2 Branch Model

The input for the branch model was considerably larger due to the number of possible combinations of input states. While it was infeasible to check all of the states evaluated in the trunk model, we were able to test all of the transitions that started or ended in the 1-bit modified and 2-bit modified sets and involved a baseline state. As previously mentioned, we also incorporated the false negative states. Finally the test states included the set of baseline transitions that were observed during testing of the dam. The test transitions were generated by taking each of these pairings of sets of states and computing the Cartesian product. To reduce the number of test transitions, we did not include transitions where both states were in either the 1-bit or 2-bit modified states. Using our previous trunk model analysis, we know that these modified states will be flagged using the trunk model. The resulting number of possible combinations is very large, the different parings of sets were generated as separate files to facilitate checking the test transitions in smaller batches. The findings from the trunk analysis model can also be used to inform how the transitions are generated so as to focus our analysis on how well the branch model complements the inability of the trunk model to contextualise states. Therefore, we prioritise transitions to and from the false negative states of the trunk analysis and the baseline states. In this way, we can target the analysis of how well the branch detects transitions from states that the trunk model recognises into potentially dangerous modified states. We also tested transitions between all of the false negative states to examine if an attacker can cause the system to enter a sequence of invalid states while avoiding detection by our models. Similarly, we created transitions between all of the baseline states to evaluate if our model could detect transitions occurring between valid baseline states that we have not observed transitions between during our testing of the dam. This was created by creating transitions between each of the 34 baseline states and then subtracting the baseline transitions to give 1054 fabricated baseline transitions ($34^2 - 102$ baseline transitions).

The breakdown of the pairings in the test transition dataset is shown in Table 5.1. The key omissions from the test set are transitions between the 1 and 2-bit modified states. These were not included as they would make the test data set too large to feasibly evaluate, and most of these states have already been flagged by our trunk model. However, those that have not, the

false negatives of the trunk, are included in our evaluation consisting of the recognised states from the 1, 2, 3 and 4-bit modified sets.

Start Set	End Set	Number of Transitions
Baseline	Baseline	1054
Baseline	Trunk false negatives	1768
Trunk false negatives	Baseline	1768
Trunk false negatives	Trunk false negatives	2704
Baseline	1-bit modified	9214
1-bit modified	Baseline	9214
Baseline	2-bits modified	99654
2-bits modified	Baseline	99654
Trunk false negatives	1-bit modified	14092
1-bit Modified	Trunk false negatives	14092
Trunk false negatives	2-bits modified	152412
2-bits modified	Trunk false negatives	152412

Table 5.1: Table showing the composition of the test transitions set, summing to 561,752. Trunk false negatives consists to the states that were not flagged during the evaluation of the trunk model.

The set of test transitions was then used as input into the branch model in a similar way to the manner used to test the trunk model. The results of this evaluation are shown in Fig. 5.13. As can be seen, this model does not perform as strongly as the trunk model. It recognises a significant portion of the transitions that were tested causing a high number of false negatives.

The branch model struggles to correctly identify transitions originating in an invalid state. This is in part due to the branch model not being intended for use in isolation since it assumes that the start state is a value and searches for the end state from there. This untethering of the start state dramatically increases the number of false negatives, as it is easier for the state to find a valid path when it begins at an invalid start state. Evidence for this can be seen in the breakdown of state results by the type of state that the transition began in, as shown in Table 5.2. When viewed this way it is clear that our approach works far better when the start state of the system is valid, as is the case in the trunk model.

When the start state is invalid, a dramatic drop in performance can be observed when the end state is considered valid. We can confirm this by grouping the results according to the type of their end state. This gives the results in Table 5.3, which shows that when the end state was invalid (was not recognised by the trunk model), the branch model performed with 99.55% accuracy. When the end state is valid, however, the branch model performs less reliably, identifying transitions ending in baseline states as valid 51% of the time and those transitions ending in trunk false negative as valid 55.42% of the time.

By calculating the difference between the start and end state IDs, we can identify trends in the tag changes that are not being flagged and which values change the most often. A graph of these frequencies is shown in Fig. 5.14. Fig. It indicates that the most frequently incorrectly

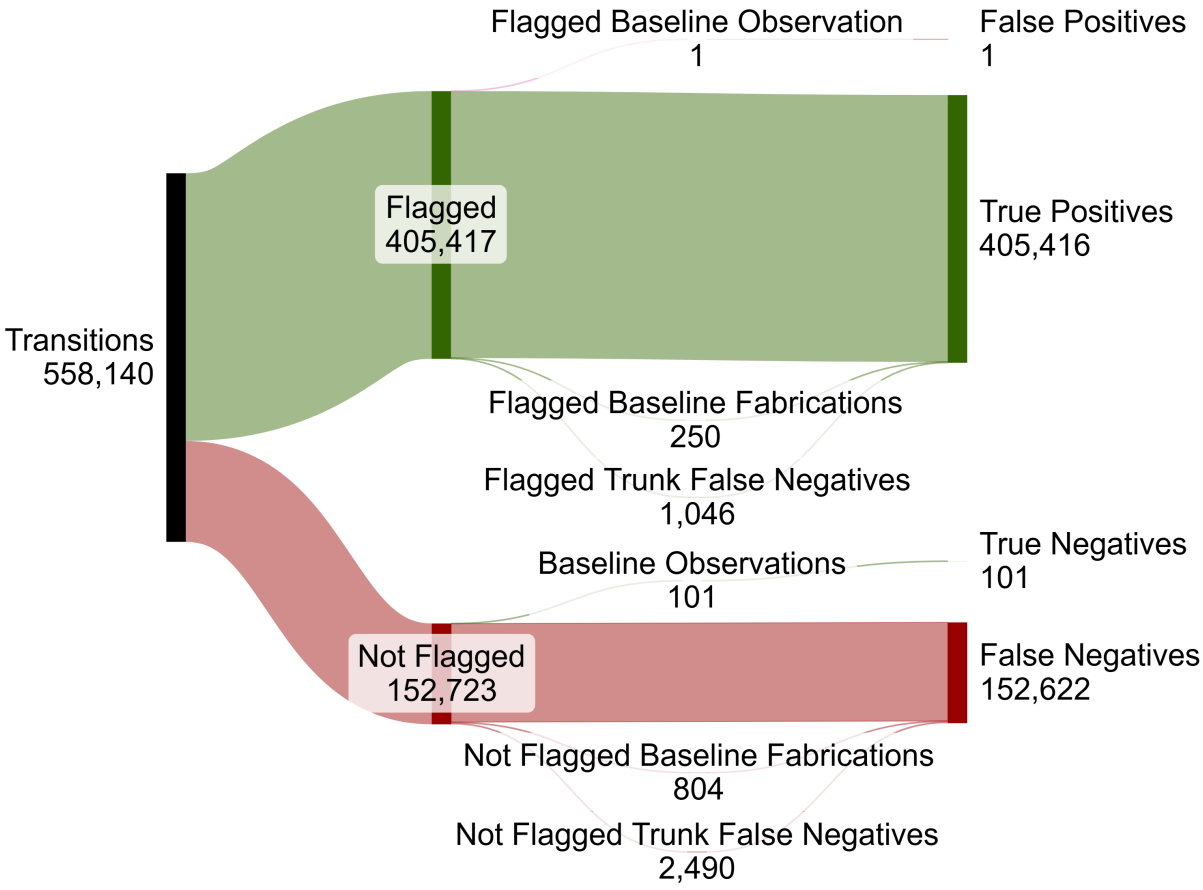


Figure 5.13: A Sankey diagram of the inputs and results of the evaluation of the trunk model. The diagram is read from left to right. The left side shows the bank of all transitions. Moving right shows the number of transitions that were flagged as anomalies or not flagged by the model and therefore recognised as normal behaviours. These groups are then categorised into true positives, true negatives, false positives and false negatives based on whether the model's output matches their true classification.

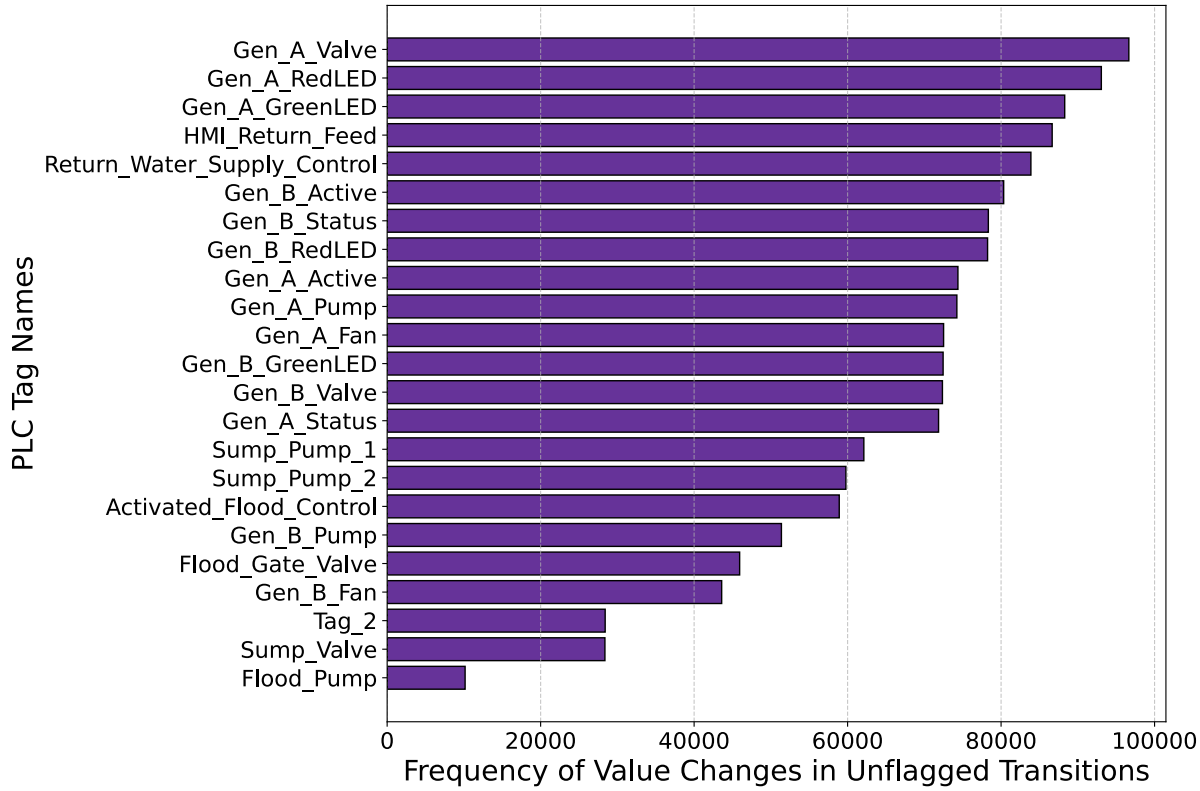


Figure 5.14: Visualisation of how often different tag value changes occured in the transitions that were not flagged during the branch model evaluation. The most frequent tags are generator PLC tags, with the exception of `Return_Water_Supply_Control` and `HMI_Return_Feed` which are, respectively, a control PLC and HMI tags

Type of Start State	Transition Result	Result Count	Percentage
Baseline	Valid	2586	2.31%
Baseline	Invalid	109206	97.69%
Trunk false negative	Valid	4688	97.54%
Trunk false negative	Invalid	1590	2.46%
Other Modified	Valid	145938	53.00%
Other Modified	Invalid	129434	47.00%

Table 5.2: Aggregated totals for transition results grouped by the type of the start state. Transitions originating from baseline states perform significantly better than those originating from modified states. A perfect score for the baseline group would be 102 valid results to 112,302 invalid, or less than 0.001% valid. All of the transitions originating from a state outside of the baseline group are anomalies so a perfect score for those rows would be 100% invalid results.

Type of End State	Status	Count	Percentage
Baseline	Valid	57330	51.00%
Baseline	Invalid	55074	49.00%
Trunk false negative	Valid	95748	55.42%
Trunk false negative	Invalid	77034	44.58%
Other Modified	Valid	491	0.45%
Other Modified	Invalid	108377	99.55%

Table 5.3: Aggregated totals for transition results grouped by the type of the end state. Transitions ending in baseline states perform significantly worse than those ending in modified states. A perfect score for the baseline group would be 102 valid results to 112,302 invalid, or less than 0.001% valid. All of the transitions finishing in a state outside of the baseline group are anomalies, so a perfect score for those rows would be 100% invalid results.

identified states contain generator state transitions. This is likely because an invalid configuration of the system is homogenised by the design of our code. To illustrate this, we refer to the simplified excerpt from the branch model shown in Fig. 5.15. Our generator process is designed to perform valid assignments to tag values reflecting changes in the system state. When a generator's "Active" tag changes state, all of the associated tags are changed to reflect this, as is the case in the PLC code. However, when invalid start states occur, this design is undermined as the design does not check for if the start state was invalid before making the new variable assignments. To correct this, a change to the model would be required to check that the previous state of the system is valid before changing state. Such a check could be performed on line 2 of Fig. 5.15, checking that the values of the generator's associated values are correct for an inactive state before allowing it to become active. No such check is performed in the PLC code, however the representation of the process's logic is still consistent.

Our branch model failed to recognise one transition from our set of baseline states. We can examine the states involved in this transition and compare them with our model to understand why this normal behaviour has been flagged. The relevant tags and their values are presented in Table 5.4. As can be seen the transition is not recognised because it violates our underlying

```
1     if
2     :: Gen_A_Active == true -> goto Gen_A_On
3     fi;
4         /* Turn on Generator A */
5         Gen_A_On: skip
6
7         /* Assign generator values appropriately */
8         d_step {
9                 Gen_A_Valve = true;
10                Gen_A_RedLED = false;
11                Gen_A_GreenLED = true;
12        ...
13                Gen_B_RedLED = true;
14        }
```

Figure 5.15: Excerpt from the branch model that allows invalid generator configurations to be recognised. Since the model doesn't check the start state configuration, an invalid start state consisting of both LEDs being on can transition into a valid configuration through these d_step sequences, which are executed for any configuration of Generator A and B activations.

assumption that the operator can only perform two actions in the time between the system gathering new data. In the start state, Generator A is active, but the control PLC has yet to receive this from the generator PLC. In the end state this is also the case. Since the control PLC will certainly have received and updated its state within the transition time window, the only explanation for this state occurring is that the Generator has been de-activated and then reactivated during the transition window. Through these two actions, the control PLC would receive the update that the generator was active, then another that it was inactive, resetting Gen_A_Status to 0. Completing this sequence of operations takes two actions from the operator. This occurs alongside the deactivation of Generator B, which requires an additional action from the operator. There are, therefore, not enough actions to complete this sequence of events, so the transition is flagged as anomalous. Since this is in the set of baseline behaviours, the model could be modified to allow three operator actions to occur, allowing the model to recognise this type of activity as normal. However, our assumption that an operator should only perform two actions per transition seems reasonable. This is an extreme example of system activity since it requires the operator to turn generators on and off again rapidly within the short timespan between system state observations. It may be better for the system to flag this type of behaviour to ensure that system operators are aware of its occurrence.

The intention when developing this model was to identify invalid transitions between valid states. Such a transition would indicate a rapid change behaviour that is inconsistent with the regular operation of the system, even if it begins and ends with the system in a normal operating state. Examining the groupings of Tables 5.2 and 5.3 suggests that these states are among the least likely to be identified since they begin and end in baseline states. As shown in Fig. 5.13, the

branch model was able to identify some of these states are anomalous. Of the 1054 fabricated baseline states, 250 were identified as anomalous, with 804 false negatives. While all of these 250 transitions are transitions that would otherwise not have been identified, allowing 804 false negatives demonstrates that the model is not performing well in this regard. Similar to the explanation for false negatives in the trunk model, this could be because these transitions are possible in the dam and have not been observed in testing. There are relatively few inputs that the operator can provide to modify the system state, and the length of the system's operational loop is relatively short in terms of the number of distinct states involved. Therefore, it is certainly likely that a significant number of these transitions can indeed occur with only two actions on the operator's part. This type of branch model may, therefore, perform better in an environment with a longer loop of system behaviours where the process proceeds through a sequence of stages.

	Gen_A_Active	**Gen_A_Status**	**Gen_B_Active**	**Gen_B_Status**
Start state	1	0	1	1
End state	1	0	0	1

Table 5.4: A table showing the baseline transition that wasn't recognised by our branch model. The transition shows that the Generator A is active in both states but that in both states the control PLC doesn't have this data. This implies that Generator A has been turned off and then turned back on during the transition. Additionally, Generator B has been turned off. The transition therefore requires three operator actions which is not permitted within our model.

5.5 Discussion

The overall performance of our approach is shown in Fig. 5.16. The trunk model detects anomalies in many of the anomalous states involved in the anomalous transitions that the branch model doesn't detect. This means that despite the transition being recognised by the branch model, the trunk model will ensure that the behaviour is flagged as anomalous. In this way, the trunk model mitigates the relative weakness of the branch model for this type of transition. Conversely, the branch model can improve the performance of using the trunk model alone by identifying 1,968 transitions between states that the trunk model incorrectly recognises as normal operational states.

5.5.1 Model Classification Performance Metrics

A summary table of the results of the two models and their combined performance is provided in Fig. 5.5. We use these values to calculate the precision, recall, accuracy and F1-score of each model and the two models combined. These are metrics commonly used in the evaluation of classification systems, particularly in the field of machine learning. The calculated metrics are presented in Table 5.6.

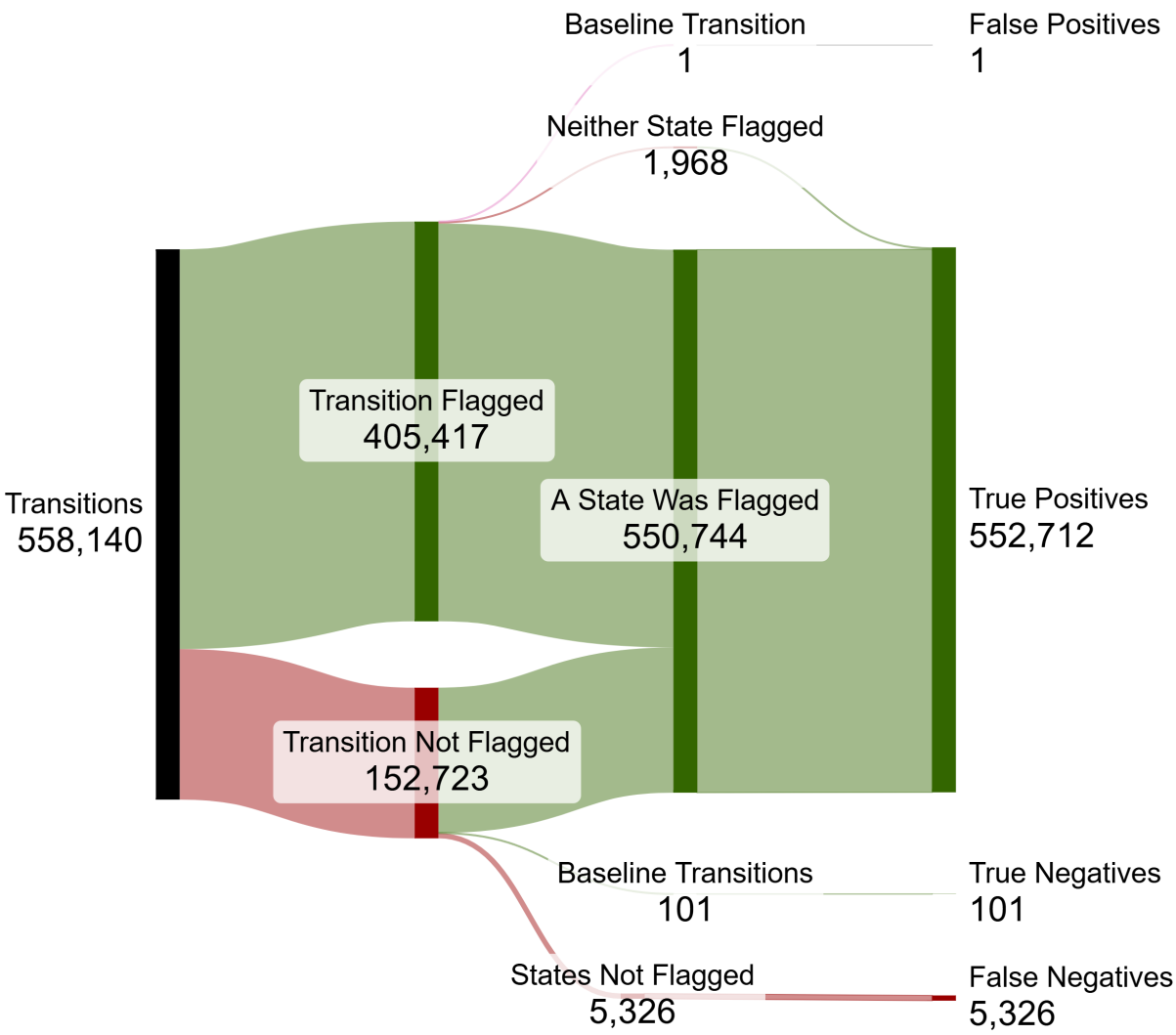


Figure 5.16: A Sankey diagram showing the improved performance gained by using the trunk and branch models together. The trunk model mitigates the weakness of the branch model by identifying and flagging invalid start states. The branch model is able to improve on the performance of the trunk model alone by identifying 1,968 anomalous transitions consisting of states that the trunk model did not identify.

Model	True Positives	False Positives	True Negatives	False Negatives
Trunk	121,530	0	34	52
Branch	495,416	1	101	152,622
Combined	552,712	1	101	5,326

Table 5.5: Summary of the total number of true positives (correctly identified anomalies, true negatives (correctly identified normal behaviours), false positives (normal behaviours incorrectly identified as anomalous) and false negatives (anomalies incorrectly identified as normal behaviours) for the trunk and branch models, along with totals for when the models are used together.

Category	Precision	Recall	F1-Score	Accuracy
Trunk	100%	99.96%	99.98%	99.96%
Branch	99.9998%	72.65%	84.16%	72.66%
Combined	99.9998%	99.05%	99.52%	99.05%

Table 5.6: Individual precision, recall, F-score and accuracy scores for trunk and branch models when used individually and when used together.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

Precision is a metric that measures the ratio of true positives to all positives. It is a signifier of how reliable a positive result is of an anomaly. Our approach scores very highly in precision, close to 100%, since almost every positive result was indeed an anomaly.

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

Recall measures how many of the total anomalies were correctly flagged as anomalies. The trunk model performs extremely well in this regard, again almost 100%, since almost all anomalies were correctly flagged. By comparison, the branch model scores considerably worse, 72.65%, because even though it correctly identified a large number of anomalies, it also allowed a large number of anomaly transitions to pass without being identified.

F1-score and accuracy are both metrics used to evaluate the overall performance of the classifier, considering each of the four outputs.

$$\text{Accuracy} = \frac{\text{True Positives (TP)} + \text{True Negatives (TN)}}{\text{Total Number of Samples}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Accuracy measures how often the classifier is correct. It is the measure of true positives and true negatives out of the set of all inputs. This is useful in a balanced dataset where there are an equal number of positive and negative cases or in the case where false positives and false negatives are considered equally as costly. However, in an imbalanced dataset, it can be

easily skewed. Our dataset is imbalanced since 99% of the states in it are anomalous, therefore a system that predicts an anomaly 100% of the time will score 99% accuracy since it weights all errors equally. In these cases, an alternative metric is better suited to weigh the classifier's performance more evenly. The trunk model and the combined approach score well in accuracy, with scores of 99.96% and 99.05%, respectively. The branch model scores considerably lower with a score of 72.66%, indicative of the number of false negatives present.

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \times \frac{\frac{\text{TP}}{\text{TP+FP}} \times \frac{\text{TP}}{\text{TP+FN}}}{\frac{\text{TP}}{\text{TP+FP}} + \frac{\text{TP}}{\text{TP+FN}}}$$

The F1-score is a specific variation of the general F-score metric. F-scores are calculated using precision and recall to better quantify a system's performance, particularly in cases where balancing false positives (FP) and false negatives (FN) is critical. The parameter β in the general F_β-score represents the relative weighting of precision to recall. F1-score uses a $\beta = 1$, assigning equal weight to precision and recall to compute the harmonic mean of the two metrics.

In our test data, we have very few true negative cases and instances of correctly identified normal behaviour. However, it is critical to the functionality of our system that a viable approach correctly distinguishes these true negatives from the large set of anomalies. With the F1-score, if our system failed to recognise any of these true negatives, it would result in an F1-score of 0%. Conversely, if it failed to identify any anomalies, it would also score %, making it a better metric for evaluating our approach. The trunk and combined approaches once again score highly in this metric with 99.98% and 99.52%. The branch model scored lower in F1-score with a score of 84.16%, a higher score than its accuracy metric due to the F1-score weighing its ability to classify almost all of the the true negatives correctly.

5.5.2 Model Integration

In our testing, we found that the analysis performed by SPIN could not initially keep pace with the rate at which data was being retrieved from the system. Two modifications to our design were used to mitigate this to an acceptable level. Firstly, a multithreaded approach was implemented where the retrieved states were pushed to a thread-safe queue to be processed by worker threads. Each worker thread can access a model template in its own working directory, embedding the state values before compiling the model and executing SPIN. Using this approach, we achieved an average search time of 7.9 seconds for the trunk model and 2.4 seconds for the branch model. The difference in these times is due to the increased complexity and scope of the trunk model. In our testing, for each transition, we had to search the trunk model for each state and the branch model for the transition between them; it took an average of 18.2 seconds for a thread to process a transition this way. This is still much greater than the rate at which new data is received (approximately once per second), so further efficiency improvements are required.

The key enabler of our approach is the use of memoisation optimisation techniques. Memo-

isation refers to the storage and retrieval of values that are computed through expensive function calls. In this case, when a new state or transition is identified, SPIN is executed as previously described. Upon completion, the value calculated, `True` for a recognised normal behaviour or `False` for an anomaly, is stored in a table alongside the identifier for the state or states in the transition. When that state or transition is next encountered, the system can retrieve the pre-computed value instead of performing another SPIN search. Using this approach significantly reduces the computational load of our approach during operation. When the system is started, the memoisation tables are empty and the rate at which new states are observed outpaces the speed at which SPIN can process them. Values are stored in a queue for processing and may not be processed until previous system data is checked. This leads to a delay in response but not the absence of a response. As the worker thread processes the observed states and appends their computed values to the memoisation table, the system stabilises, and searches are only executed for previously unobserved data. Until this happens the system is more vulnerable. In a more complex system, the time taken to achieve stabilisation may be increased; however, this approach is practical in all systems with repeated behaviours or cycles. Once this stabilisation has occurred, the time taken to identify an anomaly can be as low as 10.3 seconds in the likely scenario that the previously observed state has been previously computed.

While the results of our evaluation are promising, the model can be further refined through an iterative design process. The misclassified states can be used to identify flaws in the representation of the system. However, these changes should only be made if it can be confirmed that they are indeed anomalous. It may be that they have not been observed in testing but are theoretically possible in practice, prompting a discussion about whether or not this behaviour is intended. Therefore, refinements to the model should ideally be made jointly with the designers of the underlying system being modelled, who can determine whether identified anomalous behaviours are valid examples of normal behaviour or not.

5.6 Summary

In this chapter, we developed a specification-based approach for validating the behaviour of the PLCs within the hydroelectric dam. Model checking is used to provide a comprehensive understanding of the possible states that the dam can reach, using that as a baseline with which to compare observed behaviours.

We show how we constructed a logical representation of the PLC behaviours by inspecting their code. We provided excerpts of the Promela model and explain our process of replicating the original behaviour, written in PLC Ladder Logic. We used this transcription process to create two models of the dam, a “trunk” and a “branch” model, to assess state snapshots in different contexts. Each model also includes a representation of the operators of the dam through the integration of an HMI process that provides user inputs to the dam that can change the state.

The trunk model captures the set of all possible reachable states that the dam can reach and is used to assess whether a given set of state values is valid. The branch model has a variable start state and can be used to check if the dam has a valid execution path between given start and end states.

We discussed how we integrated the two developed models into the DT framework developed in Chapter 4. Each snapshot of state data is embedded into the trunk model through the use of placeholder values that are overwritten. The instantiated model, containing the state data, is then checked with SPIN to determine if that set of state data is valid. If the state is valid, SPIN will return a counterexample containing a sequence of system states that would allow the system to reach that state. We used the presence of this counterexample to determine that the state is valid. If no counterexample is found, the state is unreachable through normal operation of the dam and we conclude that it has occurred due to an anomaly that should be investigated. Each system snapshot is also checked for validity against the previously collected system snapshot by instantiating an instance of the branch model with the previous and current snapshots. SPIN then checks this branch model instance to determine if the system could transition between these two observed states during the time interval between them being gathered.

We evaluated our approach by generating a large set of anomalous variations of observed system states. By modifying the baseline states, we created a set of the most subtle anomalous states that an attacker might try to put the system into.

The trunk model was evaluated against a dataset of 34 baseline states and 122,182 variations of those baseline states with up to 4 modified tag values. Of the resulting 122,216 state variations checked, SPIN recognised all the baseline states as normal state configurations, with only 52 anomaly states that were incorrectly recognised as normal.

A subset of the anomaly test states was used with the baseline states to create a set of test transitions between pairs of these states. This created a test dataset of 558,038 anomalous transitions and 102 baseline transitions that had been observed in the dam previously. Checking this dataset revealed that the branch model recognised all but one of the baseline transitions, it performed weaker than the trunk model recognising 152,622 anomalous transitions. We analysed these anomalous transitions and determined that the model performed poorly specifically on the set of transitions that began in an anomalous state. However, since the trunk model performs strongly at identifying these anomalous start states, this weakness is almost eliminated when we combine the two models. When we analysed the performance of the branch model with the results of the trunk model to identify the number of transitions that contained states and transitions that neither approach identified. This resulted in only 5,326 anomalous system transitions that were not flagged correctly and a single incorrectly identified baseline transition.

Metrics for evaluating the performance of classification systems were used to assess the effectiveness of our approach. Each of our models was very effective at identifying baseline behaviours, with each scoring almost 100% precision. Our trunk model also performed

very strongly in recall, a measure of how well the system identified anomalies, where it scored 99.96%. Though our branch model was the weaker of the two models, it still achieved a recall score of 72.65%. We calculated F1-score and accuracy for the two models to evaluate overall performance. Our trunk model achieved an accuracy score of 99.96%, while our branch model achieved an accuracy score of 72.66% due to the high number of falsely recognised transitions. Overall our approach scored 99.05% accuracy. Since our dataset contains predominately anomalous behaviour, F1-score is a better metric for evaluating overall performance than accuracy as it corrects the class imbalance better. Our trunk model scored 99.98% F1-score, the branch model scored 84.16% and overall, the F1-score of our approach is 99.52%. While our evaluation cannot practically evaluate the entire set of observable behaviours, the results of our evaluation on a subset of them suggest that this technique can deliver highly reliable identification of system anomalies.

Chapter 6

Modelling Digital Twin Networks using Alloy

In this chapter, we propose using formal methods to construct representations of computer networks. We model the networks that comprise the digital thread connecting the DT with its physical counterpart, which can then be used to search for patterns of behaviour consistent with threats that affect DTs. Identifying these threats can expose design vulnerabilities early in the development process when preventative measures can most easily be introduced.

6.1 Alloy

We use Alloy to model the structure and design of the digital thread within a DT using the techniques outlined in Section 3.4. Alloy consists of two parts. The first is the Alloy specification language that is used to describe structures and how they interact. The Alloy Analyzer is used to assemble the structures that have been described, combining them using the rules and constraints contained within the model to generate instances. The generated instances satisfy the given constraints and show an example of an arrangement that can occur in the modelled system. The search for examples can be focussed through the use of predicates that narrow the search for counterexamples that violate the statements made in the predicates. We utilise the native support for traces that were added in Alloy 6 to allow for our model to consist of a sequence of states and for us to check properties over the length of this sequence. This provides a powerfully expressive language for describing structural designs that change over time.

As Alloy uses first-order logic to create its models, checking the validity of assertions for all cases is undecidable [115]. However, Alloy is not designed to provide proof of a property, rather to provide refutation. Given a bounded scope within which to search, the alloy analyzer will provide sound analysis that is guaranteed to find a counterexample if one exists within the given scope.

The process used to perform this type of analysis follows a standard iterative model-checking

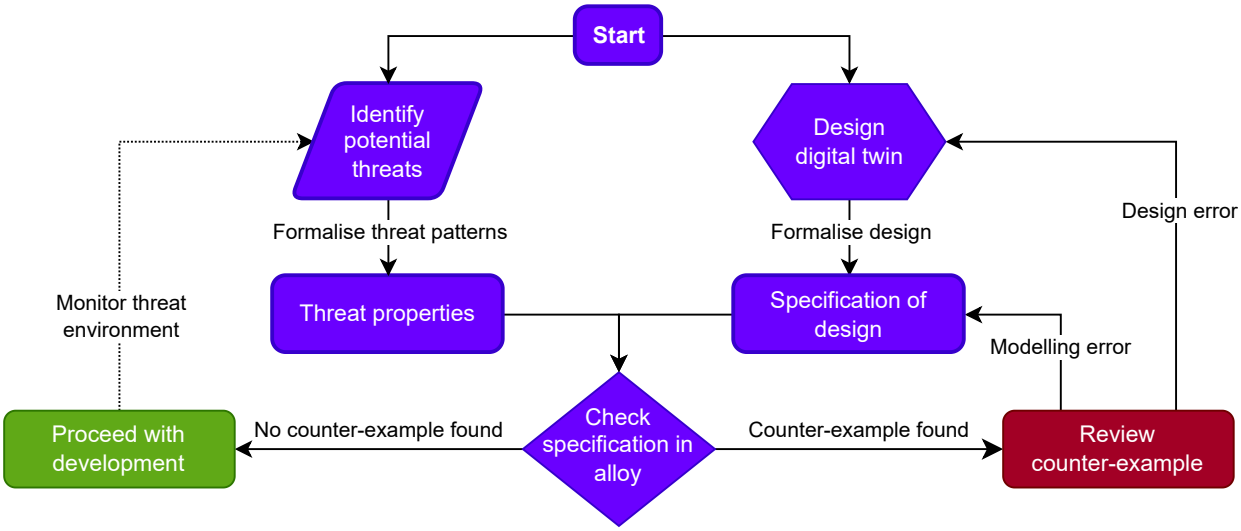


Figure 6.1: The alloy modelling process described as a flowchart. An alloy specification of a DT design is iteratively checked for properties representing threats identified in its operational environment.

process. Fig. 6.1 illustrates how this technique can be used during the design of a DT. The approach begins with an initial design for a DT. During the design process, the types of threats that exist in the environment to which the DT will be deployed are considered. In order to assess if sufficient mitigations exist within the given model, a specification of the design is created in Alloy that contains the key components of the system and their interactions. Using this specification, threat properties are constructed to represent the patterns of behaviour that need to be checked within the design. The alloy analyzer then checks the specification for the presence of the given threat properties. If a counterexample is found, it is either due to a flaw in the design or an error in the design specification. If it is a modelling error, the specification is corrected, and the analysis is performed again. Similarly, if the counterexample exposes a flaw in the system's design, then additional mitigations are added to the design, a new specification representing the changes is created, and the analysis is performed again. This process continues until no counterexamples are found. At this point, if the specification represents the behaviours of the design, then the process will have provided additional assurance that the model is robust to the threats modelled within the threat properties, and development can proceed.

6.1.1 Modelling Network Security in Alloy

Alloy has previously been applied to modeling Cyber-Physical Systems (CPS) and Internet of Things (IoT) networks [136, 138]. In [138], an Alloy model is presented that represents IoT subsystems communicating over a network of channels. This approach was later extended to verify cyber-security standards in [136].

The method of representing a network of devices in these works is intuitive and broadly applicable to various challenges. We adopt a similar approach in the early models of this chapter. However, while [136, 138] focus on the interactions between IoT subsystems in an unstructured network—demonstrating eavesdropping and identity-faking attacks—our approach shifts the focus towards the network architecture and its role in security.

Our objective is to better understand how the network architecture should be designed to securely connect two specific systems: the DT and the ICS. As we will show in this chapter we achieve this by developing and refining our model to explicitly represent the network's structure and the exchange of datasets within it. Additionally, our approach makes the presence of a threat actor explicit, defining them as an entity within the network. This entity possesses a known dataset and a set of possible actions to expand its knowledge and influence within the system.

6.2 Introduction to the Alloy Specification Language

In Section 6.7 we construct alloy models of the connective digital thread within a DT network. In this section we will examine several models of increasing complexity that were created in the process of developing these more advanced models.

Since the technologies within the DT thread vary depending upon the twinned system, our goal is to construct a modelling approach that abstracts many of these specifics by considering them from the perspective of the attacks that can be carried out on them. This will allow us to focus specifically on the key characteristics most relevant to security during the transfer of data between the physical and virtual spaces.

The models in this chapter reflect some behaviours of a computer network – implementing them in Alloy is a process of iterative refinement to make them better reflect the behaviours of the network while staying within the bounds of computability. Since this process affects how these models have been designed, we will present them in sequences that demonstrate this refinement process. In Section 6.3 we refine an initial modelling approach through a series of improvements to the representation of data and devices in the model. In Section 6.4, these refinements are used as a foundation to expand the concept of message passing which is then used to illustrate IP and ARP spoofing attacks in a DT network. Finally, we implement significant advancements to the previous models that allow it to be applied to check for vulnerabilities in our DT framework developed in Chapter 4 and 5.

6.3 Initial Model Development

We start with the specification shown in Fig. 6.2, which contains a simple Device signature specified in Alloy 5 that represents a device within a network. Below the signature name is a

```
1 sig Device {
2         // A device has least one connection
3     channel: some Device
4 } {
5         // Devices cannot be directly connected to themselves
6     this not in channel
7 }
8
9 run show {} for 3 Device
```

Figure 6.2: Initial alloy model of devices linked with channel relations.

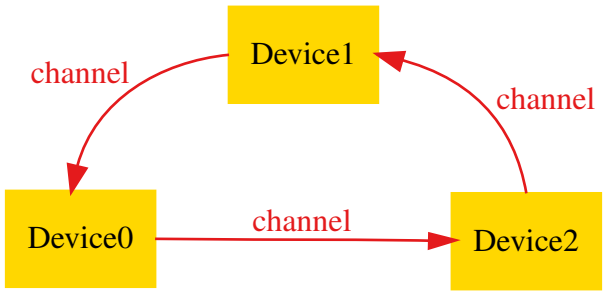


Figure 6.3: The minimum alloy instance that can be generated by executing the model in Fig. 6.2

single relation channel that specifies that each atom of the Device signature is related to at least one other device. This is followed by a single signature fact that states that device atoms cannot be related to themselves through the channel relation. This simple model allows us to generate simple representations of a network of devices. By executing the run command shown in the listing and providing a scope of 3 Devices, we can generate the instance shown in Fig. 6.3. The figure shows that Alloy has identified an instance with the three devices connected in a loop of directional channels and is just one example of a satisfying instance for our given constraints. Multiple examples of 2 and 3 devices, as well as an empty network containing no devices, satisfy the constraints.

This simple model contains a method of representing how devices within a network are connected. In our representation, we need to be able to associate devices with the data that they know. To examine how these devices communicate across these channels, we introduce a new Data signature representing data that is being shared between devices.

Data must be able to move through the network, as devices send or receive it. Consequently, the relations used to represent data must also be able to move. There are a few different ways to represent this in Alloy, we will consider some of these in this chapter. The original method of representing a sequence of events in Alloy is to use the `ordering` [1] module. The module puts the elements of a signature into an ordered list and provides some functions to enable facts that restrict the order in which these elements can be arranged. Using this module we

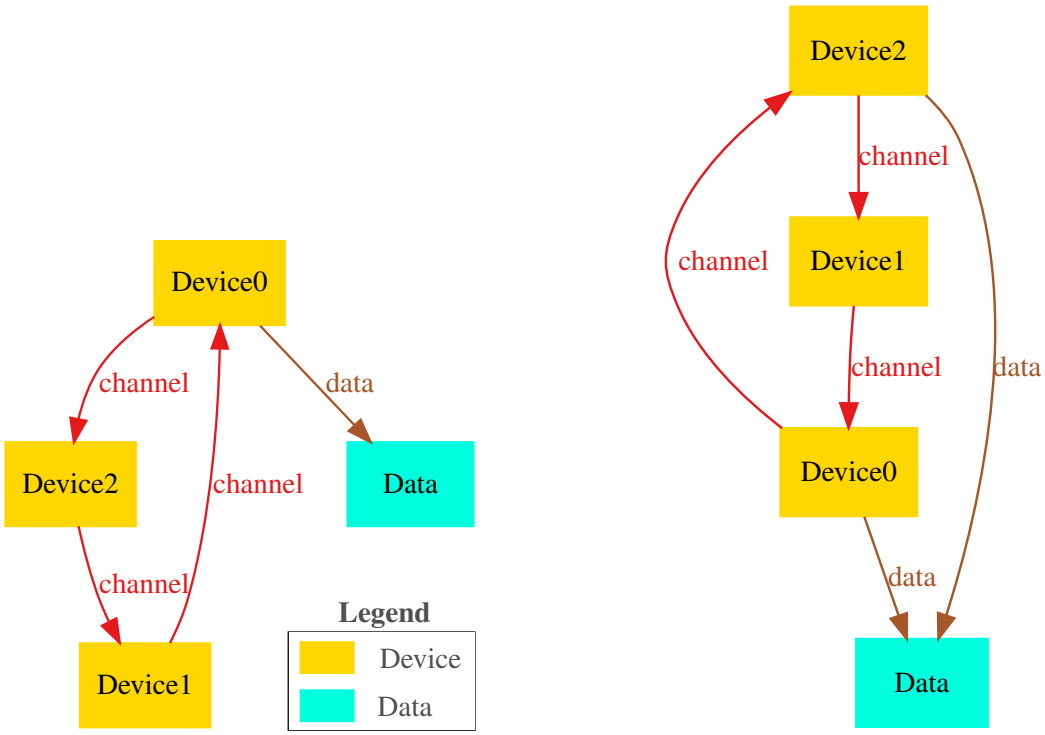
[1] `https://alloy.readthedocs.io/en/latest/modules/ordering.html`

```
1  open util/ordering[State]
2
3  sig Data {}
4  sig Device {}
5  sig State {
6     // The set of data known by each device in each state
7     data:    set Device → set Data,
8     // A device has least one connection
9     channel: set Device → set Device
10 }{
11    // Devices do not have channels directly to themselves
12    all d: Device | d not in d.channel
13    // Each device has an indirect channel to every other device
14    all d, d': Device | d' in d.^channel
15 }
16
17 fact{
18    // Network channels remain consistent between states
19    all s: State, s':s.next |
20       s.channel = s'.channel
21
22    // Previously known data is retained in future states
23    all s: State, s': s.next, device: Device |
24       s.data[device] in s'.data[device]
25
26    // Devices share all data on all connections between states
27    all s: State, s': s.next,
28    sender: Device, receiver: s.channel[sender] |
29       s'.data[receiver] = s.data[receiver] + s.data[sender]
30 }
```

Figure 6.4: Devices model enhanced with data sharing.

create an updated version of our model, shown in Fig. 6.4. The new State signature contains all of the relations, now specified as mappings between atoms. `State.data` is a mapping from each Device atom to all of the Data atoms that the device knows in that state. `State.channel` plays a similar role to the previous `channel` relation, although it is a relation within the `State` signature mapping each device to the devices that it can communicate with. In line 1, we use the ordering module on the State signature to force Alloy to arrange the state atoms into an order. Using facts within the State signature, we restrict channels. In line 14 we use the transitive closure operator `^` to access the set of devices accessible by applying the channel relation recursively. This gives us all the devices connected to a device, both directly and indirectly. This allows us to enforce that for every device, `d`, every other device, `d'`, must be present within that set.

The other facts after line 17 specify how relations can change between states, this is how we specific temporal behaviours in the model. The facts starting on line 23 and line 27 show how the



(a) Device 0 knows some data.

(b) The Data has been shared with Device2.

Figure 6.5: A two-state trace showing an instance generated by the Alloy Analyzer when running the model shown in Fig. 6.4.

mappings within `State` can be used. For example, `s.data[device]` refers to the set of data that the device knows in State s. In those two facts, we specify that devices retain knowledge of data that they've previously learned and that devices share that data with all devices that they are connected to. Instances can be generated as before by using the run command. However, this time, the instances consist of multiple States. By projecting the relations in State over the atoms in the instance, we see the sequence of states as shown in Fig. 6.5. Viewing the states and the ordering in which they're presented allows us to observe the data being shared between devices. In the initial state of the trace shown in Fig 6.5 (a) we can observe that Device0 has a piece of Data that it knows and a channel to Device2. After the transition, that Data has been shared with Device2 since both devices now have data relations to it.

With this foundation we can now introduce our first mechanism for modelling intrusions into the network, compromised devices. When a device in a network becomes compromised it can be used to attack or undermine the operation of other devices. Compromised devices might attempt to take control of another device by sharing an exploit with them, or they might share modified or incorrect data to try and get them to spread that data with (and thus compromise) other systems. We can take the first step to model this by expanding the `State` signature to label devices that are compromised and data that are malicious as shown in Fig. 6.6. Constraining the behaviour in this way requires two components. On line 18, we keep track of all compromised devices

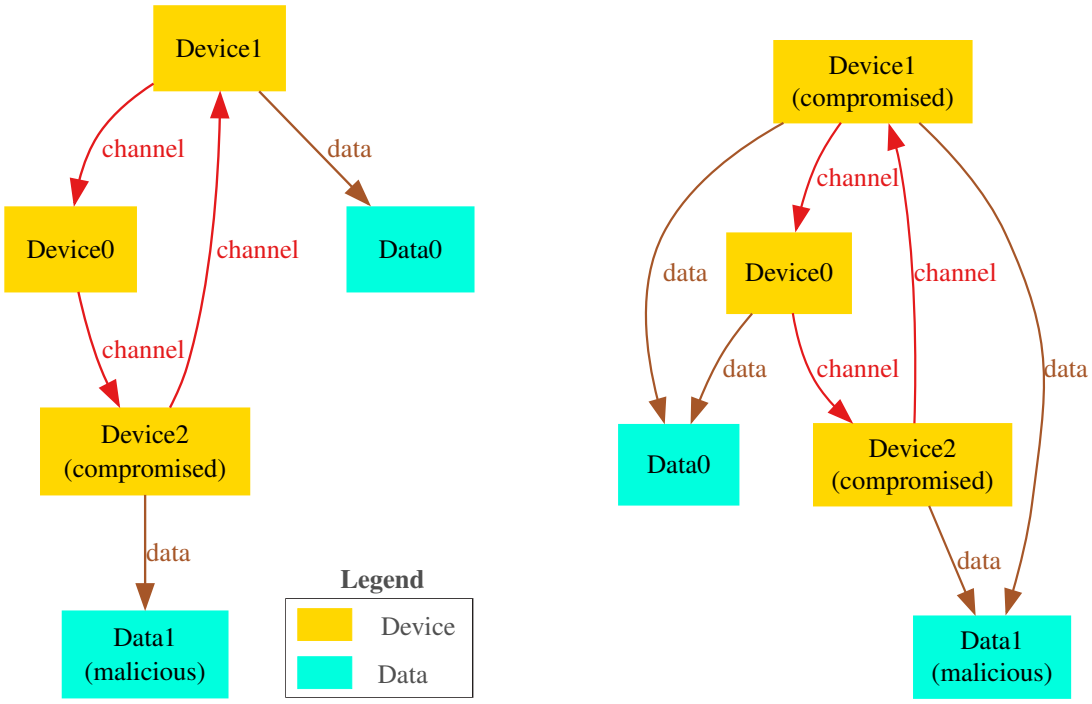
```
1  sig State {
2     ...
3     compromised: set Device,
4     malicious:   set Data
5  }{
6     // Devices that interact with malicious data are compromised
7     all datum: malicious, d: Device |
8        datum in data[d] implies d in compromised
9     // All compromised devices have accessed some malicious data
10    all device: compromised | some datum: data[device] |
11       datum in malicious
12 }
13 fact{
14    ...
15    // Compromised devices are retained in future states.
16    all s: State, s':s.next |
17       s.compromised in s'.compromised ∧
18       s.malicious = s'.malicious
19 }
20 pred init(s: State) {
21    // Must include some malicious data
22    #s.malicious = 1
23    // Not all devices can be compromised
24    not s.compromised = Device
25 }
```

Figure 6.6: A listing from an alloy model showing the modifications to State that enable compromised devices to share malicious data and compromise other devices.

and malicious data, ensuring that previously compromised devices remain compromised in the future. This is enforced through the use of the `in` operator to ensure that the set of devices that were compromised in a previous state is at least a subset of the devices in the next state. On line 20 we constrain the set of malicious data to remain the same, so in this model compromised devices do not create more malicious data.

The signature facts on lines 7 and 10 of `State` define how malicious data compromises devices that interact with it. The first fact enforces that any device that knows malicious `Data` is in `State.compromised`. This alone may seem sufficient to constrain the model. However, because we use implication in the previous fact, we must cover instances where the conditional in the expression `datum in data[d]` is false, in this case devices that have not interacted with malicious data can also be compromised. Therefore, we prevent Alloy from compromising devices that have not interacted with malicious data.

The created model represents a basic system where a malicious intruder can compromise other systems directly by sharing malicious data with other devices. An example of what an instance of this model looks like is shown in Fig. 6.7. It shows how the movement of data from a compromised device to an uncompromised device. In the initial state, shown in Fig. 6.7 (a)



(a) Device 2 sends malicious Data to Device 1.

(b) Device 1 receives the malicious Data and becomes compromised. Device 0 remains unaffected.

Figure 6.7: A two-state trace of the model from Fig 6.6 showing malicious data compromising Device 2.

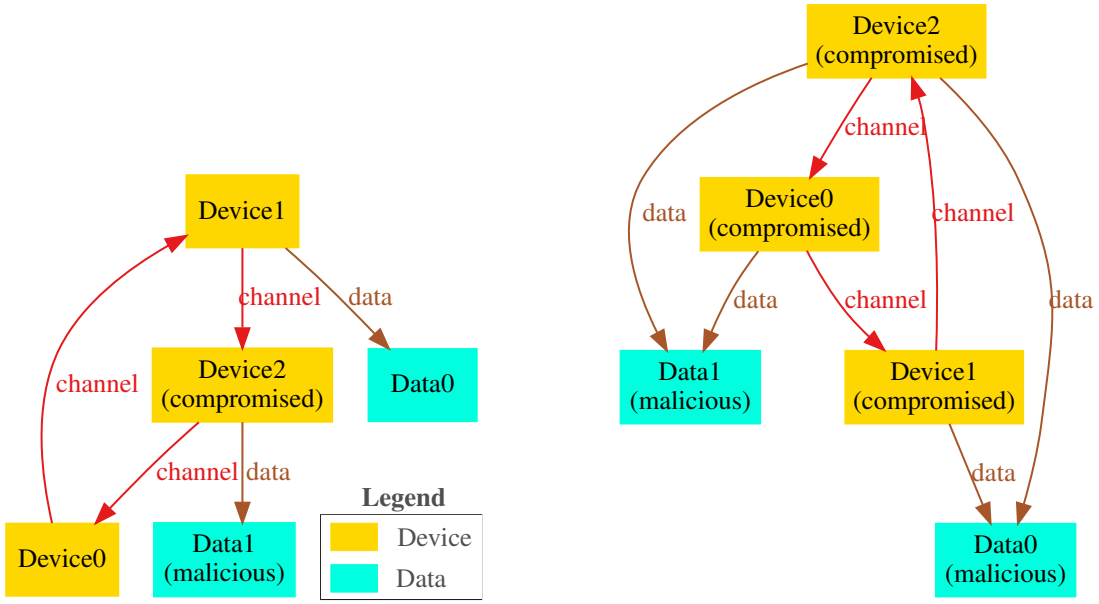
Device2 is compromised, and has created a piece of malicious data. It has an outgoing network channel to Device1 and we can observe, in Fig. 6.7 (b), that after the transition this data has been shared with Device1 potentially undermining its operation and compromising it.

DTs rely on a network of devices sharing data to form a connected thread; if one device in that thread becomes compromised, the data all the way up the thread is undermined. To examine the propagating impact of this, our model should enable compromised devices in the network to turn clean data into malicious data. The changes shown in Fig. 6.8 modify the code in Fig. 6.6 to address this. The signature fact on line 10 of Fig. 6.6 is modified to reflect an aggressive attacker that compromises all data that it interacts with. The fact on line 13 in Fig. 6.6 removes the restriction that maintains the amount of malicious data in the system, enabling more malicious data to be created.

When testing our Alloy model, the behaviour shown in Fig. 6.8 can be seen. When an uncompromised device becomes compromised in the same transition that it sends clean data, it causes that clean data to become malicious. The clean data that has incorrectly been labelled as malicious then makes the receiver of that data compromised, despite not having interacted with malicious data. While those devices would eventually become compromised, the model skips the intermediate steps and propagates the effect too quickly. Additionally, it also backwards propagates in such a way that all devices that had interacted with that data before it became

```
sig State { ... }{
   ...
    // Replaces previous constraint (some > all)
    // All data that interacts with compromised devices is malicious
    all device: compromised | all datum: data[device] |
       datum in malicious
}
fact{
   ...
    all s: State, s': s.next |
       s.compromised in s'.compromised ∧
       // The amount of malicious data can increase between states
       s.malicious in s'.malicious
}
```

Figure 6.8: An updated State signature that enables compromised devices to turn data they interact with malicious.



(a) Clean data, Data 0, being sent to compromised Device 2.

(b) Device 2 receives Data 0, Device 1 is now compromised.

Figure 6.9: A two-state trace demonstrating the need for changes to the modelling of data in Fig. 6.8. Device 1 receives no data but becomes compromised via the sent Data 0.

malicious will be compromised. This issue highlights the need for a more advanced means of modelling the data within a system so that the location of a piece of data moving through the network is tracked independently from the list of devices that know and have interacted with that data.

6.4 State Transition Model

With the basic concepts of alloy and our modelling approach demonstrated, we now introduce an updated second model that builds upon it. In this version, we use the features provided in Alloy 6 to expand the potential of the model with mutable signatures and fields and the increased functionality of temporal operators in the specification of the model's behaviours and properties [2]. The addition of the var keyword allows for the specification of a mutable signature or field that can change between states. A generated instance with these mutable elements consists of a trace that can be traversed in a similar, but natively supported, manner that allows for the State signatures in our previous model to be observed in sequence. By adding these mutable signatures we can remove the need to store time-variable relations in a separate State signature and instead move those relations into the signatures. As we will show in this section, this substantially improves the expressiveness of the model, while also enhancing readability of complex relations as the functionality of the State relations is handled within each variable signature. Additionally, there are now more powerful temporal operators that can be used to enhance facts. The model is no longer restricted to only utilise functions provided by the ordering module that consists of operations providing access to the order dictated by the ordered signature. Instead we can modify conditional facts directly with operators such as eventually , always, until , and historically to give a more nuanced specification of how the networked devices interact.

Using the new variable signatures of Alloy 6, along with the updated temporal operators, the model can be restructured as shown in Fig. 6.10. The updated version retains all of the capabilities of its predecessor while being smaller and more readable since all of the facts are now signature facts. It also includes the addition of the Data. location field that tracks data as it moves across the network. Attaching the labelling of malicious data and compromised devices to this allows for a more useful representation of network vulnerability in our model, as devices can receive a piece of data and inadvertently pass it on to a compromised device without becoming compromised themselves.

This model does not yet help answer questions about how the system is vulnerable. We can introduce further elements to model how data moves through the network and so investigate ways in which this process may be interfered with. We first focus on IP messaging and, to do this, introduce the Message signature as shown in Fig. 6.11. To reduce the size of the model, a stream of IP packets is condensed into a single IP message. To focus on the movement of data, we primarily include the source and destination header fields. Other flags can readily be integrated here but would increase the size of the model, limiting the depth of the analysis. When focussing on ARP and IP spoofing attacks these other flags are less important for our analysis so they are omitted.

We add an identity to the Device signature (equivalent to a network address) so that the

[2] https://alloytools.org/alloy6.html

```
enum Status { Clean, Compromised, Malicious }

some var sig Device {
    var status: Status,
    channel: some Device,
    var data: set Data,
}{
    // No self channels
    this not in channel
    // Devices remember what data they know
    data in data'

     // Devices are compromised if:
    status = Compromised iff {
        // the device has always been compromised,
        historically status = Compromised
        // or the device became compromised in a previous state,
        or before status = Compromised
        // or some compromised data was received
        or some datum: data |
            once (datum.status = Malicious and datum.location = this)
    }
}

some var sig Data {
    var status: Status,
    var location: one Device
}{
     // Next location is at most one channel move from the current one
    location' in location + location.channel
    // Devices know data that is located at them
    this in location.data

    // Data is malicious if
    status = Malicious iff {
        // it was previously malicious
        before status = Malicious
        // or if it has ever been at a compromised device
        or some device: Device |
            once (location = device and device.status = Compromised)
    }
}
```

Figure 6.10: An Alloy listing showing mutable signatures for Device and Data. A ’ refers to that field in the next state of the trace.

```
// Messages carry data across the network
some var sig Message {
    source: one Device,
    destination: one Device,
    payload: one Data,
    var location: one Device,
}{
    location' in location + location.channel

    // Messages actually have to go somewhere.
    not source = destination

    // Payload data is only compromised if
    payload.status = Malicious iff {
          // if it was compromised in the previous state.
        before payload.status = Compromised
          // or the message is at a compromised location
        or once location.status = Compromised
    }
    // Messages deliver payload and stop at their destination
    location = destination implies {
           payload in location.data
           location' = location
    }  else {
        // Otherwise they must move along a channel
        all topology: Topology | location' in location.channel
          // They are always routed to their destination
        eventually location = destination
    }
}
```

Figure 6.11: An Alloy listing showing the addition of the mutable signature for an IP Message that carries data through the network. Data.location becomes Message.location and the rules for compromising data are modified accordingly.

devices are distinguishable and identifiable by each other.

To illustrate our process further and demonstrate its utility, we model a simple architecture of a DT in an ICS. We define instances to represent the components of the physical system, the DT and the infrastructure used to connect them using the abstract device signature shown in Fig. 6.12. Our model now represents the other components with signatures for Data, messages, channels and network topology. It uses extensions of the Device signature for different types of devices, such as the DT, PLCs, network switches and attackers. Messages transport data across the network; during each state transition, the location of each message will change to model the process of transporting data across the digital thread. Using this basic model, we can then represent the techniques that may be employed by threat actors to interfere with these mechanisms.

```
some abstract sig Device {
  var status: Status,      // Type: enum {Clean, Compromised}
  identity: Device,        // Public identity (IP address)
  created: set Data,       // Data created by device
  var accepted: set Data, // Data consumed by this device
  var messages: set Message // Messages currently at this device
}{
  // All devices are connected to at least one channel
  some chan: Channel | this in chan.connected

  accepted in accepted' // Accepted retained between states

  // Reciprocates device.messages and message.location
  all message: messages | message.location = this

  // Only switches can be part of multiple channels
  all disj chan1, chan2: Channel {
    this in chan1.connected & chan2.connected implies
        this in Switch
  }

  // Device compromise conditions
  status = Compromised iff {
    this in Attacker
    or before status = Compromised
    or some data: accepted | before data.status = Compromised
  }

  // If device is clean it has never been compromised.
  status = Clean implies historically status = Clean

  // Non-attacker devices have their own identity
  this not in Attacker implies identity = this

  // All accepted data is from a message sent to this device
  all data: accepted | some message: Message {
    message.payload = data
    and once message.location = this
    and data not in created
  }
}
```

Figure 6.12: An Alloy listing showing the abstract signature that the Attacker, PLC, Switch and DigitalTwin signatures inherit from.

Each Device has a status to indicate whether they have interacted with malicious data or devices. They have a set of data they have created, which can be transported to another device in a Message. At each step of a satisfying instance or counterexample, each device has a set of messages which are currently being sent or received by the device. Finally, each device has a set of messages that it has received over the network and has accepted. Accepted data is data that the device trusts and which may then be used within the device's operation. If a device accepts compromised data, its status will become compromised.

```
some sig Message {
  source: one Device,          // Sender identity
  destination: one Device,     // Where the message is going
  signature: one Device,       // True sender
  payload: one Data,           // Data contained in message
  var location: one Device     // Current location of data
}{
  not source = destination     // Messages go somewhere
  source not in Switch         // Switches don't send data
  destination not in Switch    // Switches aren't sent data

  // No messages are intentionally sent to the attacker
  destination not in Attacker

  // Payloads are only compromised at a compromised location
  // or if they were compromised in the previous state.
  payload.status = Compromised iff {
    location.status = Compromised
    or before payload.status = Compromised
  }

  // Messages move until they reach a valid destination
  // Where they remain and their payload is accepted
  location.identity = destination implies {
    payload in location.accepted
    location' = location
  }      else not location' = location

  // Routing: all Messages eventually reach a valid destination
  eventually location.identity = destination

  // Messages move along channels unless at a valid destination
  all network: Topology |
    location -> location' in network.connections
    or location.identity = destination
}
```

Figure 6.13: An Alloy listing showing the Message signature used to transport data between devices across the channels of the network.

In this model, we represent attackers as an extension of the device signature to enable them to behave differently from non-attacker devices. While the identity of a non-attacker device is fixed, we make additions to the Device to provide them with an identity . A device's identity is equivalent to their network address and denotes and is how it identifies itself to the other devices on the network. We use the extension to the device signature to allow the identity relation of the attacker to point to any device to which it is connected via a channel. We use other extensions to the device signature to represent DTs and PLCs. In our scenario, the PLCs are the components of the physical device that send data to the DT about the process they are controlling and their own internal state. A final extension to the device signature represents network switches. These are the only devices which can be connected to more than one channel. The Message signature, shown in Fig. 6.13, moves data across the network. A message has a source, current location, destination and data payload but no status, as this is represented by the status of its payload, an instance of the data signature. A message moves through the network along channels until it reaches its intended destination. Once it reaches its destination, its data is accepted by the destination device.

6.5 Threat Analysis and Identification

Using predicates, we specify threats as sequences of behaviours that should not occur within the system. In the example shown, we demonstrate this by using a predicate for a spoofing attack. The system modelled doesn't contain a mechanism for authenticating the source or destination of the data messages, leaving it vulnerable to spoofing attacks.

In the context of DTs, we consider how a rogue IoT node could carry out a spoofing attack within the physical system. In this attack, an attacker with elevated privileges, often an insider, has either compromised an element within the physical system or deployed a new network node to the system. This node is able to intercept messages from the channels using spoofing techniques such as Address Resolution Protocol (ARP) spoofing. Once the attacker has acquired the data, they can inspect it for private information and then send a message containing modified data to the DT. Alternatively, the attacker can create entirely fabricated messages of their own to send to the DT using Internet Protocol (IP) spoofing to impersonate a legitimate end node. We show how Alloy can be used to model both types of attack.

In Alloy, the check command asks the compiler to find counterexamples that negate the given properties. We use this facility to find instances where vulnerabilities are exploited in our modelled network.

Our first property allows us to check if an attacker can acquire a message by faking its identity. This property (in comments) and its Alloy representation are shown in Fig. 6.14. In this property, we assert that all messages eventually reach their intended destination. The analyzer finds a counterexample trace which starts at the state shown in Fig. 6.15. Here a PLC

```
1    //For all messages:
2    //     the message reaches a clean destination.
3    all message: Message |
4        eventually message.location = message.destination
```

Figure 6.14: Property 1: Used to identify ARP spoofing attacks

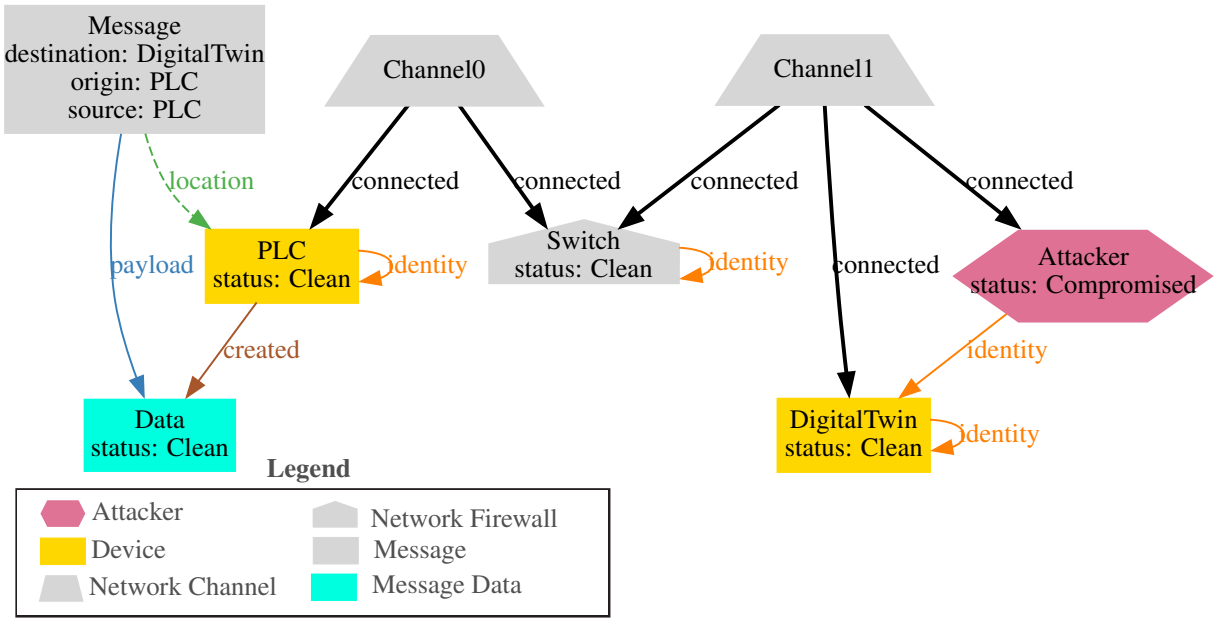


Figure 6.15: Initial state in the ARP spoofing attack trace. The PLC begins sending created data to the DT.

has created some data that it has put into a message with the DT as the intended destination.

The second state of the trace is shown in Fig. 6.16. The message has traversed the first channel and is now located at the switch connecting the channels the PLC and the DT use to communicate. However, the attacker can insert a malicious node into the channel with the DT. Using its presence on the channel, it can deceive the switch and present itself as the DT. Since both the DT and the attacker have the same identity as the destination address, they compete to receive the message, and the message can be received by either device.

The final state of the trace is shown in Fig. 6.17. The attacker has successfully spoofed the identity of the DT to the switch, causing the switch to send the attacker messages intended for the DT. The impacts of this upon the system are twofold: firstly, enabling an attacker to gain access to private system data, and secondly, preventing the DT from receiving important state from the physical system. This suggests that additional measures should be implemented within the system design to mitigate this type of spoofing attack.

By modifying the property, we can identify a different variation of the spoofing attack. In this example, we check that the DT only accepts trustworthy data. The property used to check for this and its Alloy representation are shown in Listing below.

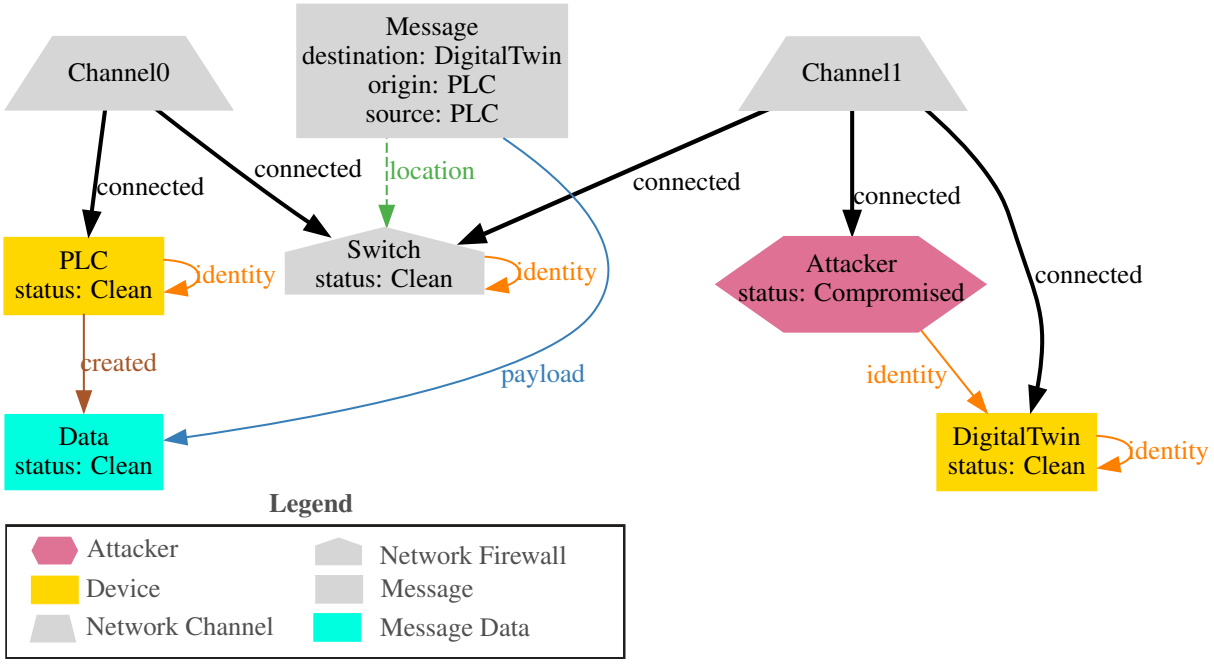


Figure 6.16: Second state in the ARP spoofing attack trace.

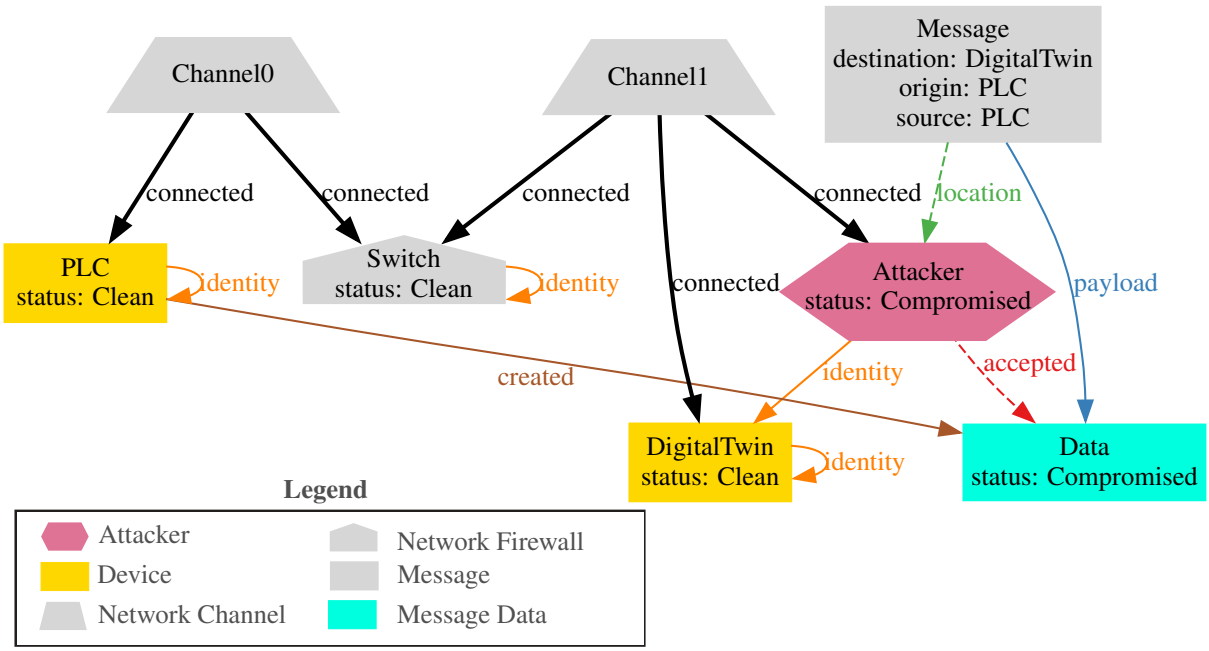


Figure 6.17: Final state in ARP spoofing attack trace. An attacker has used a rogue IoT node to spoof the DT's identity and intercept the message before it reaches the DT.

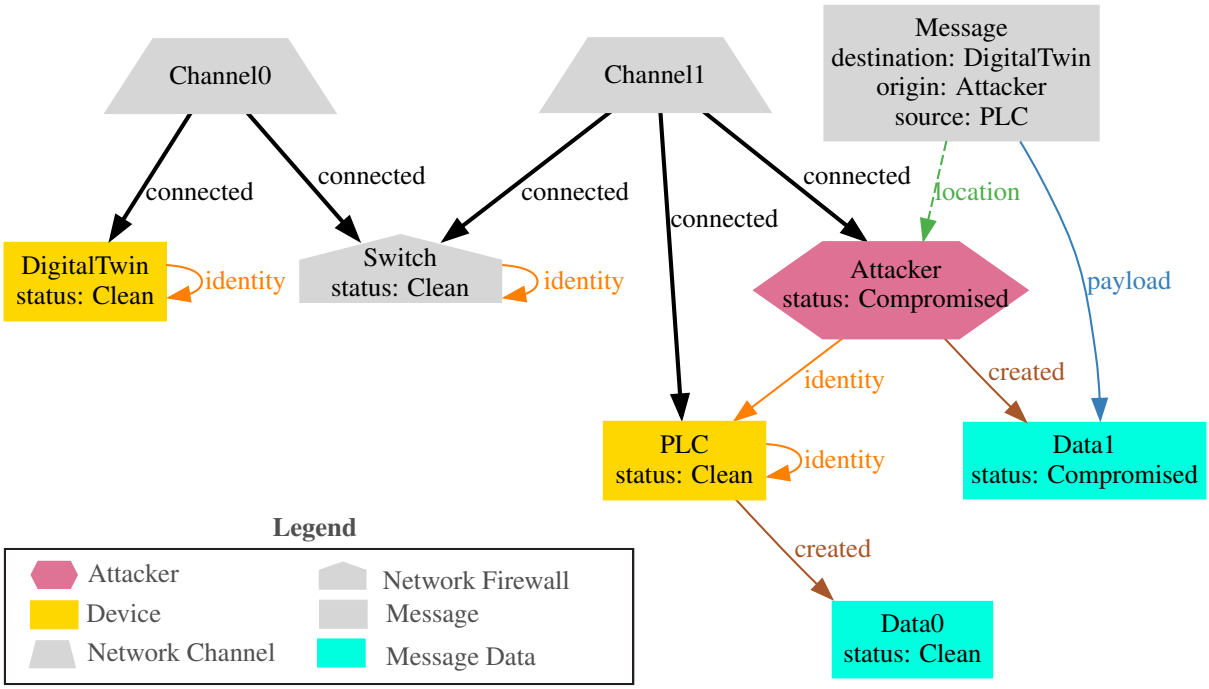


Figure 6.18: Initial state in an IP spoofing attack trace. An attacker has modified the source address of a message to send compromised data to the DT.

```
1    // All data that the DT accepts is clean.
2    always all dt: DigitalTwin, data: dt.accepted |
3        data.status = Clean
```

Figure 6.19: Property 2: Used to identify IP spoofing attacks.

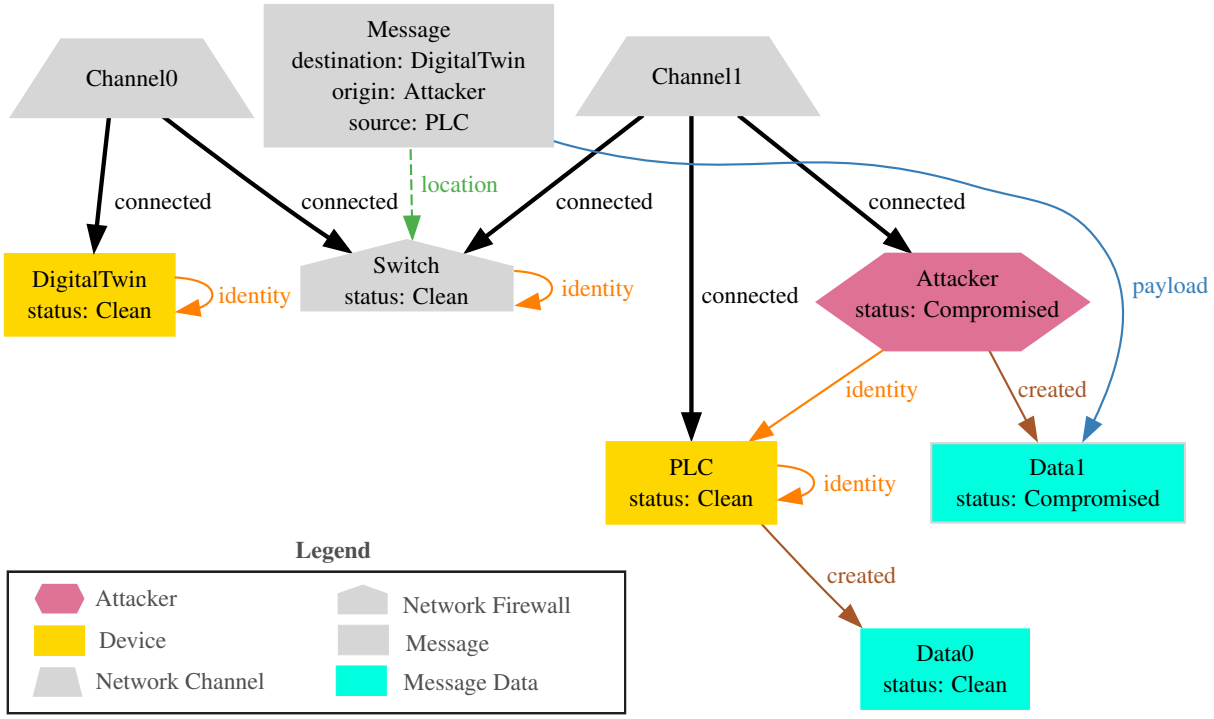


Figure 6.20: Second state in an IP spoofing attack trace.

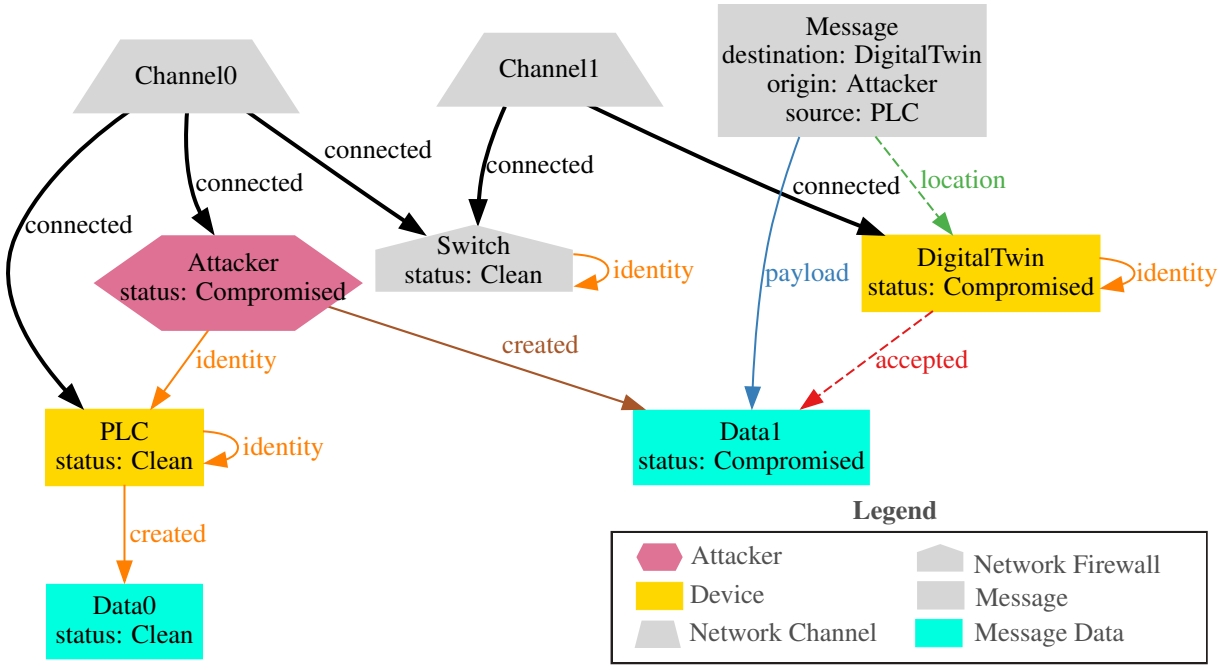


Figure 6.21: End state in an IP spoofing attack trace. The modified message is accepted by the DT, compromising the system.

In this instance, we find an attack can be carried out in the first state. As shown in Fig. 6.18, the attacker creates a message with the source identity of the PLC. This can be done in practice through the use of readily accessible IP spoofing tools that allow a user to change the source IP address on sent messages.

Through the subsequent states of the trace, we see the message pass through the switch and reach the DT, where it is accepted. Since the message originated from a device in the set of attacker devices, it compromises the DT. The existence of this trace suggests that the design of the system requires additional mitigation steps to prevent attacks of this nature. In the case of the DT, this may lead to it incorporating falsified sensor readings into its calculations.

6.6 Experimental Results

Table 6.1 shows the execution times taken to find a counterexample for the IP spoofing attack scenario. Table 6.2 shows the execution times taken when a search was executed for a complex scenario that isn't possible in the model. This approximates the maximum expected execution time to validate the specification in the model. While we do not include the number of variables or clauses or the resulting memory usage in either of the tables, these increase in line with search time. The times were recorded for a variety of scopes when running the scenario on an Apple M3 Chip with 16GB of RAM using the Glucose SAT solver [16].

Execution time increases with scope size across all search configurations. Generally, increasing the number of messages has the largest impact on execution time when an attack is

Devices	Chans	Messages	Time (ms)
5	3	1	211
10	3	1	355
15	3	1	606
5	3	2	222
10	3	2	407
15	3	2	711
5	3	3	243
10	3	3	474
15	3	3	869

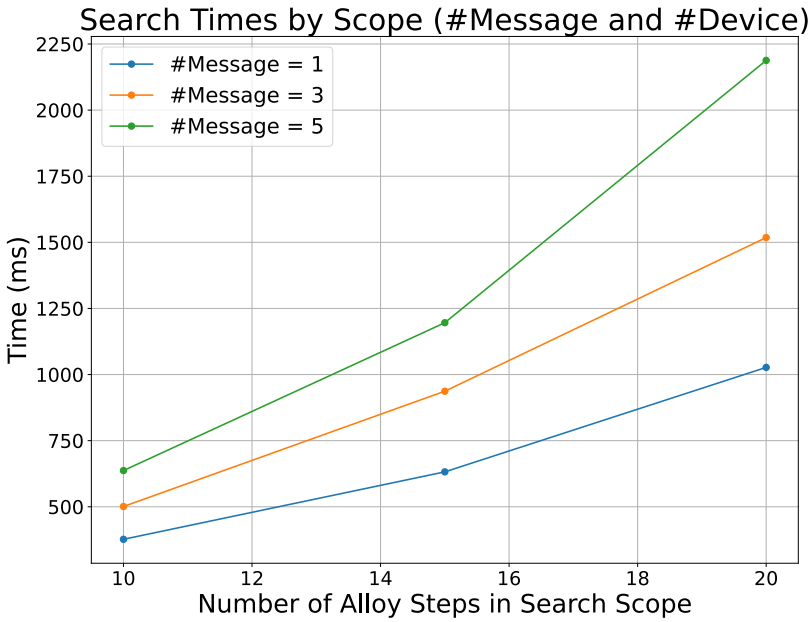
Devices	Chans	Messages	Time (ms)
10	4	1	377
15	4	1	632
20	4	1	1027
10	4	3	501
15	4	3	937
20	4	3	1518
10	4	5	637
15	4	5	1196
20	4	5	2188

Table 6.1: Execution runtimes for the IP spoofing attack under different configuration settings. In each case #Data = #Messages and scope is 10 steps, though a counterexample is found before reaching that limit. In each case, time is the median of five executions.

present, particularly in combination with a large number of devices. While the search times do increase with the size of the network, even for a moderately large number of devices, an attack is found within a few seconds running as shown in Fig. 6.22. The search is fast because the Analyzer doesn't need to finish searching the whole scope since an attack scenario can typically be identified within a few message exhanges, an example of the small scale hypothesis [115].

In scenarios where no attack is found, the search takes longer as the Analyzer searches the whole scope; in these instances, the number of steps in the search scope is the biggest factor that causes execution time to increase. A search for a simulated attack in a network of 20 devices connected by 4 channels, with 5 messages and data being shared, takes approximately 25 seconds to search all possible 10-step traces. When that same network is searched for all possible 20-step traces, the analysis takes 5.6 times as long, finishing after 2.5 minutes. Similar increases can be found at smaller network sizes, as shown in Fig. 6.23. An extreme example that pushes the limits of the analysis with a scope of 10 channels, 15 messages and 15 data illustrates impacts of the state-space explosion problem on the model with a median execution time of 40 minutes.

These results show that the approach can scale to a number of devices sufficient to model systems of a larger scale. While ICS networks and DTs can consist of hundreds of devices, for the purposes of analyzing a digital twin network not all of these devices need to be included in the scenario. A vertical representation of the systems spanning the digital thread would only require one or two of the devices at each layer of the digital thread in order to analyze the network's security between layers while also representing an attacker's ability to move laterally within a layer. Additionally, if a larger scope is required of one element within the model, for example, a longer search trace or a larger number of devices, the scope of other elements can be reduced to compensate, enabling faster search times. If a larger scope is still required after that the use of high-performance computing systems provides a viable method of further expanding



1.3

Figure 6.22: Graph of results shown in Table 6.1 showing the time taken to find counterexample in various scope configurations.

Devices	Chans	Data	Messages	Steps	Time (ms)
20	4	1	1	10	9677
20	4	1	1	13	16775
20	4	1	1	15	20631
20	4	1	1	17	31216
20	4	1	1	20	46908
20	4	3	3	10	15470
20	4	3	3	13	28701
20	4	3	3	15	40519
20	4	3	3	17	54127
20	4	3	3	20	82716
20	4	5	5	10	25872
20	4	5	5	13	49817
20	4	5	5	15	68614
20	4	5	5	17	93575
20	4	5	5	20	145102
10	10	10	10	20	85438
15	10	10	10	20	154978
20	10	10	10	20	282333
20	10	15	15	20	2383727

Table 6.2: Execution runtimes for the IP and ARP spoofing model under different scopes when no counterexample is found. In each case, time is the median of five executions.

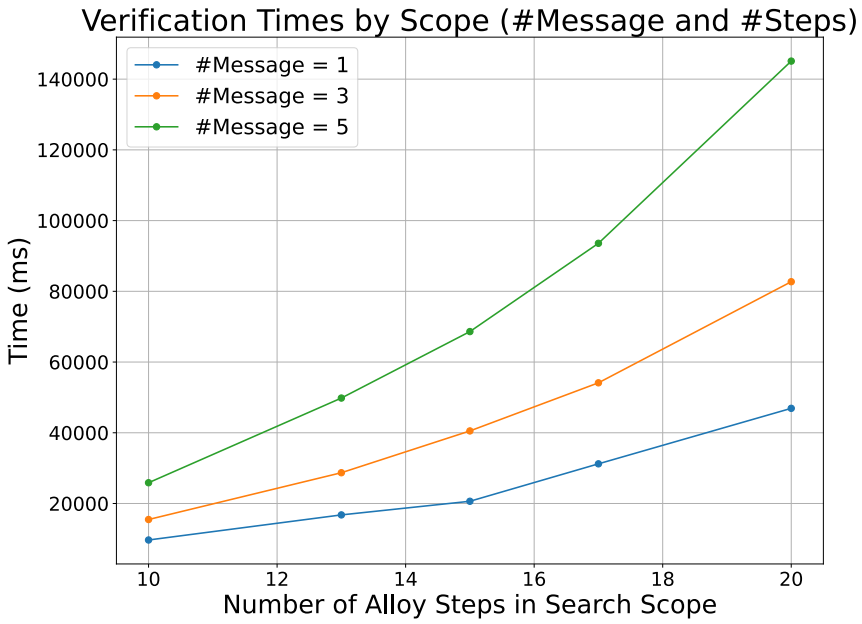


Figure 6.23: Graph of results shown in Table 6.2 where #Device = 20, #Chans = 4, and #Data = #Message. Shows the time taken to search the entire scope for a counterexample.

the size of the system that can be modelled. The increased memory would be able to contain the larger boolean matrixes that need to be generated by the Alloy Analyzer to analyse the problem at larger scopes. Additionally, the increased computation power, paired with hybrid or parallel decomposition strategies, would allow for a faster search of the given scope. Therefore while DT systems can grow to be large complex networks, the approach we present provides sufficient scalability to model the key aspects of these systems and identify design flaws therein.

6.7 Application to Hydroelectric Framework

To explore the utility of this modelling approach towards enabling the evaluation of DT design, we used it to model the hydroelectric dam developed in Chapters 4 and 5. The new model uses the concepts developed in the previous sections as a foundation; it still centres around the notions of Devices, Channels, and Messages. However, several enhancements to the model were developed to represent network configuration details and create a deeper representation of the threat model. In this section we will present the changes that were made before progressing to a demonstration of the capabilities of the new model.

6.7.1 Device Signature

We extend the Device signature to enable a more individual representation of the devices in our model and modelling attacks. The details of the device class are shown in Fig. 6.25. Devices

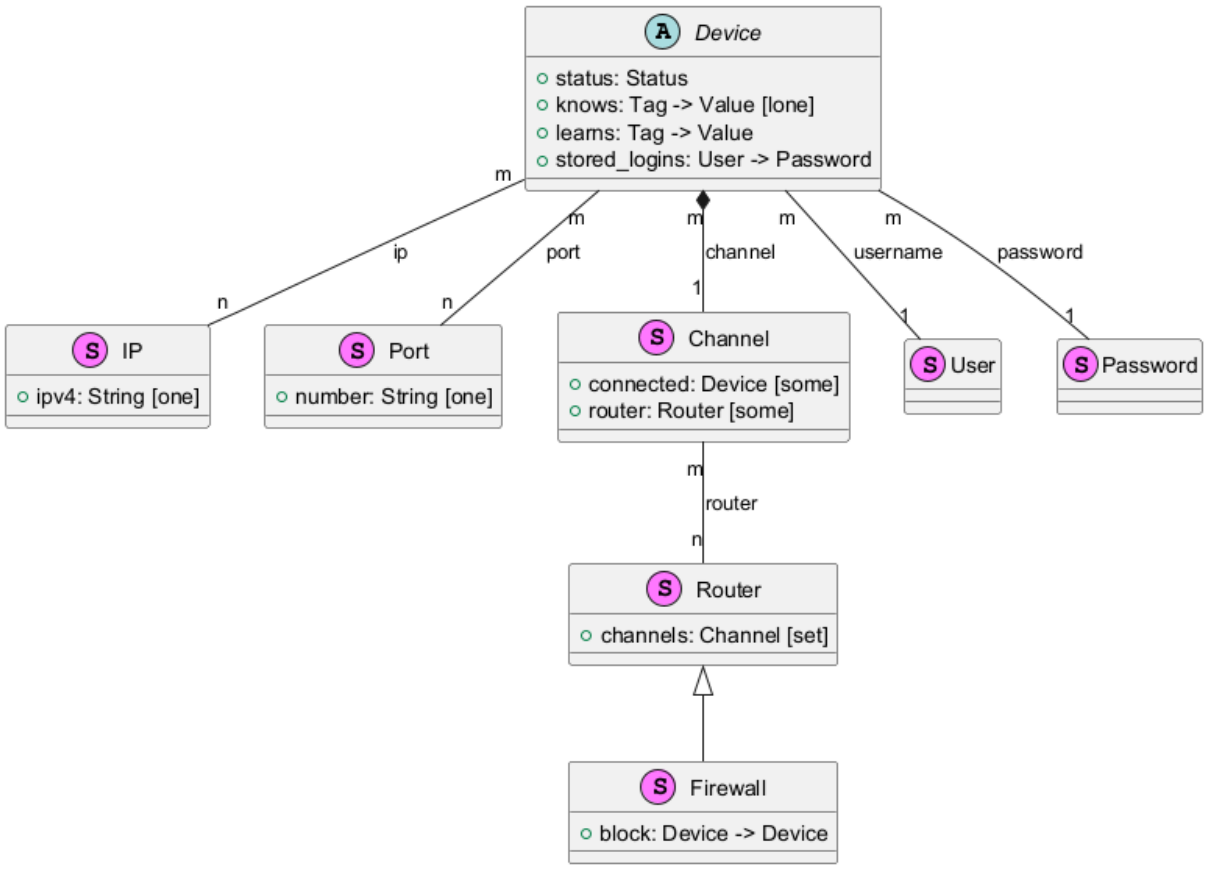


Figure 6.24: The abstract Device signature and its key relations as featured in the hydroelectric dam representation, shown in a UML diagram.

have fields for network configuration details, including IP address and port number. They also have authorisation fields, a User and a Password that can be used to access and control the device.

IP addresses and port numbers are also represented as signatures. Devices must then have a relation to at least one IP address and Port atom. Within these signatures, each has a relation to a string atom representing the device's ipv4 address in the case of the IP signature or port number in the case of the Port signature. It is through these relations that we associate an IP address and port to each device. We can constrain exactly what String is associated with each device using facts in each device's signature. Since strings are represented in Alloy as an atom with a string value enclosed, the exact value of each string does not impact the model. What matters is which string atom is involved in a relation. Our analysis of IP and Port numbers is, therefore, driven by an association to string atoms, not string values. However, the correct values are used during the analysis to ground the model in the scenario and make it more understandable.

Users and passwords are represented as signatures in the model. They are both empty signatures used to refer to what user and password combination grants access to the system. Additionally, a device has a stored_logins field that denotes a set of known Username and Password connections

6.7.2 Device Types

The representation of specific devices is significantly increased in this model through Alloy's signature system. This allows for the declaration of subsignatures of the `Device` signature. Subsignatures are made for different types of devices, as shown in Fig. 6.25. We define abstract signatures for devices with an abstract subsignature of device for PLCs.

Within the Device signature, we define signatures for the devices hosting the different components within the DT framework, specifically NodeRED, InfluxDB, Grafana, an engineering workstation, and the SPIN Model developed in Chapter 5 . We can restrict the number of instances of these signatures according to the design of the network; in the case that we're modelling, there is exactly one instance of each of these. We, therefore, constrain them with a multiplicity of **`one`**, allowing us to handle the assignment of IP addresses and port numbers within the signature facts for readability; this means they apply to all instances of that signature. If there were multiple device instances in the model, this could still be specified through facts.

PLCs are implemented as an extension of the Device signature. We create an abstract signature PLC to contain PLC-specific behaviours. This signature can then be extended, like the Device signature, to represent the Generator and Control PLCs. In our scenario model, PLCs are the only devices that can provide concrete information about the state of the dam. In the context of the previous model, they are the only devices that legitimately create Data about the state of the dam. This will be discussed further when we explain our data model in Section 6.7.5.

6.7.3 Network Connections

Fig. 6.25 also shows how devices are connected. Every device in the network is connected to a channel. As before, channels consist of sets of devices that are connected and can freely communicate; this represents a zone within the Purdue model. We use separate channels for the PLCs (together), the historian devices, the DT model and an individual workstation. Routers connect these channels, allowing for messages to flow between them. These routers also allow us to model network segmentation. While technically the whole network is connected, our routers only allow connection between each channel and those directly connected to it, creating a layered network configuration. We also use a Topology signature to track all the communication paths between devices and enable us to specify properties about the network as a whole, such as the existence of indirect communication paths between devices and that the network is fully connected.

We define a Firewall subsignature of Router, which functions as a router that can block communication between specific IP addresses. This block relation replicates a firewall's blocklist, mapping devices in the channels it is connected to so that it will prevent direct communication between them. Although blocklists in operational firewalls can selectively block messages based on the message type, in our model, a block applies directionally to all messages exchanged. Us-

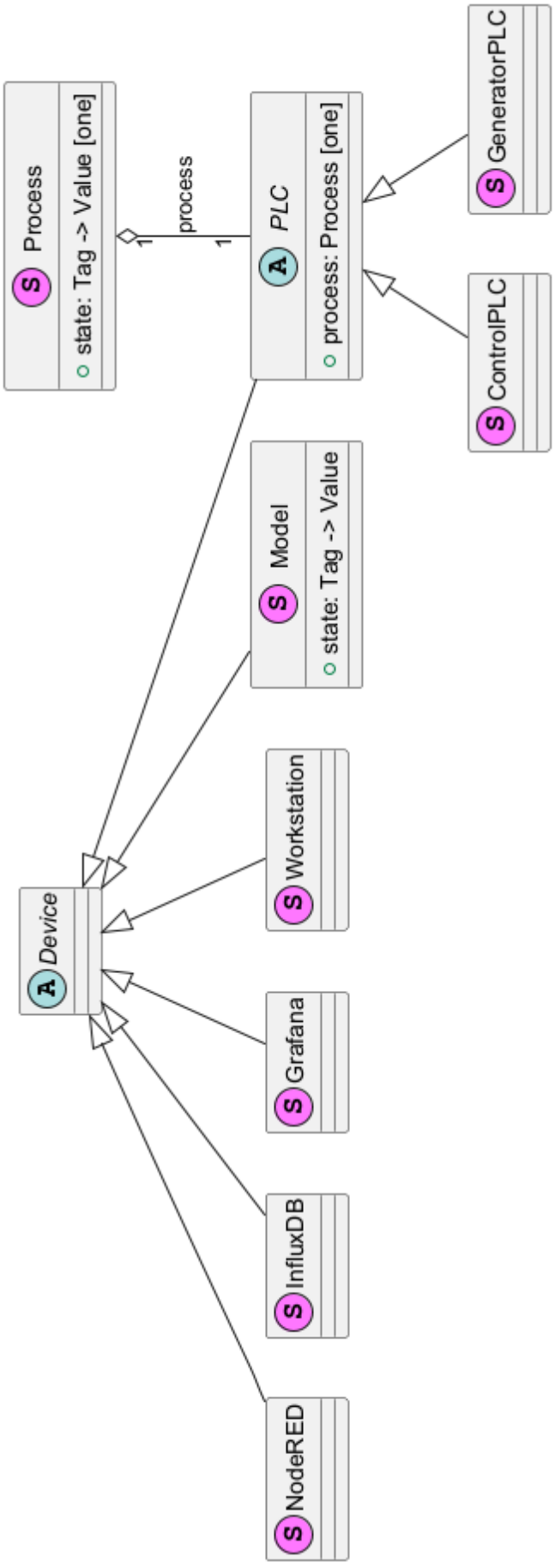


Figure 6.25: The hierarchy of Device signatures and their related subcomponents, shown in a UML diagram. Device, PLC and Actuator are abstract signatures. An Actuator is not a Device but is how the PLC interacts with the Process. Process represents the ground truth of the state of the system.

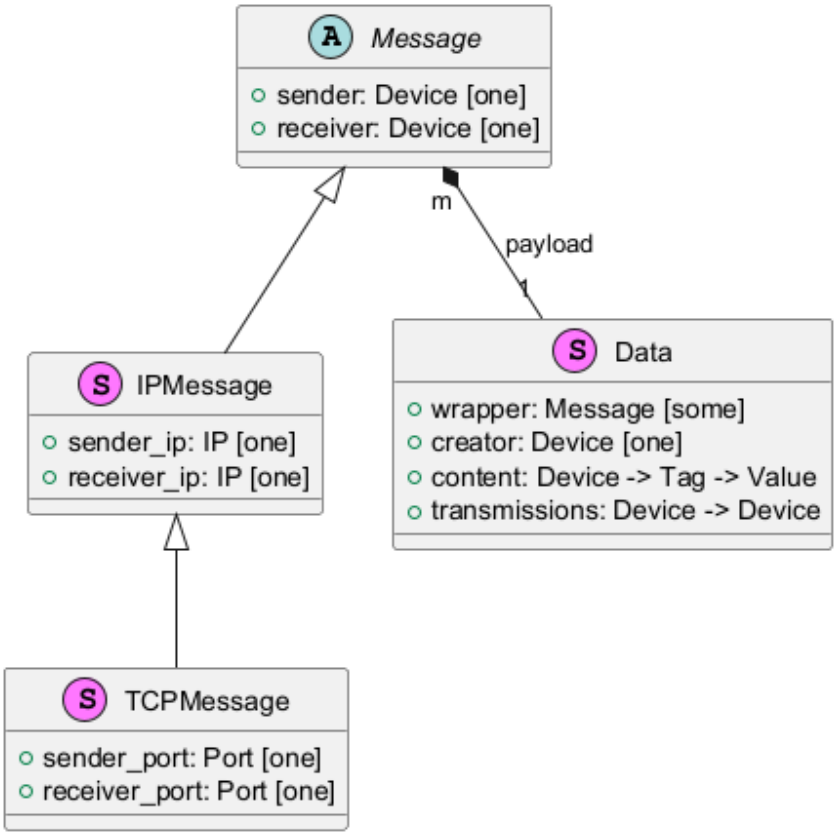


Figure 6.26: UML diagram showing the message signatures and their associated Payloads, including the Data signature.

ing firewalls in this way, we can selectively block certain components from communicating.

6.7.4 Messages

Our representation of the different types of messages in our model is shown in Fig. 6.26. We define an abstract Message signature that is similar to the previous messaging representation discussed in Section 6.4. This signature contains the key information about a message, which device sent and received it and the payload that the message contained.

The Message signature is extended with subsignatures to specify different types of messages. We specify another abstract subsignature of Message for Internet Protocol messages, then a subsignature of that IPMessage signature for Transmission Control Protocol messages. In our model, each message represents a stream of packets between the sender and receiver containing the specified payload. These subsignature messages contain the network configuration details that specify how messages move across a network. The message super-signature then reflects this in a manner that is easier to utilise in other classes that interact with exchanged messages. The message signature is also useful for checking that a sender and receiver have a valid network connection to each other.

IP messages are exchanged between a source and a destination address. For a message to be

received by a device the IP address of the receiver must be in the list of IP addresses that the receiver device has access to. For most devices, this will be the single IP address assigned to it; however, compromised devices can spoof their IP address, as shown previously. The difference in this model is that we include a representation of the mechanism by which an attacker can acquire the IP addresses that it impersonates and how it gains access to the channels in which the messages are being exchanged.

Our final message signature is the TCP message. These messages have all the fields and relations of the previous message signatures. However, since TCP packets are exchanged between applications, they also have a port number through which the host application is communicating. As with the IP addresses, most devices in our scenario will only utilise a single active port for communication. However, attacker devices can learn which ports their targets use to communicate and exploit that informaton to intercept messages or interact with the device maliciously.

6.7.5 Data

Fig. 6.26 also shows how Data is incorporated into our model as an extension of the abstract MessagePayload signature. We use this abstract signature to demonstrate how our method can be extended to allow for other types of message payloads. Our representation of Data has been expanded to express it as a dataset being exchanged through the system. To represent the content of that data set, we use a Tag-Value association to represent how variables are stored in the PLCs. We implement this by enumerating a set of *Tags* and a set of *Values* as shown in Fig. 6.27. Relations can then be constructed to map a Tag to a Value denoting the state data that each device received in the exchange of that dataset. For example, a device that owns the piece of data *Upper_Tank* $\rightarrow$ **OF** has received sensor data indicating that the Upper Tank is overflowing.

```
enum Tag {
    Upper_Tank,
    Lower_Tank,
    Generator,
    Sump_Pump,
    Sump_Valve


}
```

```
enum Value {
    ON,
    OFF,
    OF,
    H,
    M,
    L,
    UF
}
```

Figure 6.27: Alloy enums representing the PLC tags and the values that are used to express state data in the alloy model.

To better analyse the exchange of data with respect to a digital twin, we modified the way that it is represented in our model. A code listing showing these changes is shown in Fig. 6.28. Previously, a piece of data had a location and a status, each of which could change through transitions. Data is now modelled as a chain of tag value associations that flow through the

system as a single strand in each state. This reduces the size of the state-space and allows larger networks to be modelled. Since all data movement is represented in a single transition rather than one move per transition, the number of steps required to represent a network is not related to the size of the network. Encapsulating all of the data movement in a single transition also makes it easy for us to specify properties about the state of the digital thread. Since the thread is represented directly through this Data signature we can specify properties about the mapping of variables across devices, checking for instances where they are inconsistent. We will discuss this in more detail in Section 6.8.

We demonstrate how this new data signature behaves by generating the instance shown in Fig. 6.29. The transmissions relation controls the order in which devices interact with the data. Each device along this sequence of transmissions can modify the value of the data if they have been compromised. This allows us to directly represent how an attacker can interrupt the communication between the PLCs and the Model or the Grafana monitoring dashboard.

6.7.6 Threat Model

When attackers infiltrate the network, they can exploit knowledge acquired from multiple sources to move through it. To represent this, we created a signature that contains all of the information that an attacker has about the network and the devices it can influence. The signature for our Threat model is shown in Fig. 6.30.

The threat actor begins with a single controlled device. From this device the attacker is able to interact with other devices and begin to explore the network within the constraints of its topology. The attacker gains information about the network and the devices within it through actions and by gathering information stored on devices to which it has access. Its objective is to prevent the normal function of the DT system, which we define as interrupting the reception of data by the Model.

Our threat model uses an extensible action framework to represent the techniques that attackers can utilise to move through the network. The abstract class `Action` is used to represent all actions taken by the attacker. An action has an origin device from which the action is executed, and a target device that it affects. At the beginning of an instance, alloy has already generated all of the actions that will be executed within the attack. When their requirements have been satisfied, their sequential execution is represented in our model by assigning them to the `current_actions` relation where their impacts are implemented.

We categorise the actions that an attacker can use to infiltrate the network into `Recon` and `LateralMovement`. Recon actions gather data about the network. We represent the execution of a network scan that identifies active IP addresses within a network. This is done by pinging every IP address and waiting to see which devices respond. Legitimate devices will respond confirming their address and their status in order to facilitate connections and normal operation of the network. An attacker abuses this to map the layout of the network and identify targets.

```
sig Data { ...
   creator:  one Device,   // Source of data
   // Tag values for each device
   content:  Device → Tag → Value,
   // Movement between devices
   transmissions:  Device → Device,
}{
   // Clean devices must know data in order to send it
   (always creator.status = Clean) implies {
      creator.content in creator.knows
   }
   // For every tag value pair that any device knows about
   all device: Device {
      all tag: Tag, value: Value {
         tag → value in device.content implies {
            // It either knows it because it's the creator
            (device = creator or
            // Or it was sent it by a device that knows it
            some sender: Device {
               sender → device in transmissions
               tag → value in sender.content
   } } } })

   // Tying transmissions to messages for dissemination
   all send, receive: Device {
      send → receive in transmissions iff {
         some message: Message {
            this = message.payload and
            send = message.sender and
            receive = message.receiver
      } }
      // Restrictions on how data moves
      send → receive in transmissions implies {
         // Must originate with creator
         send in creator.*transmissions
         // Both sender and receiver must exchange data
         #content[send] > 0
         #content[receive] > 0
         // Data is sent faithfully if the device is Clean
         (always send.status = Clean) implies {
            content[receive] = content[send]
         }
         receive.content in receive.learns
} } }
```

Figure 6.28: The updated Data signature used to represent the data exchange across the digital thread. The content relation associates a Tag-Value pair with each device that interacts with data. The transmissions relation tracks the message exchanges that include this data.

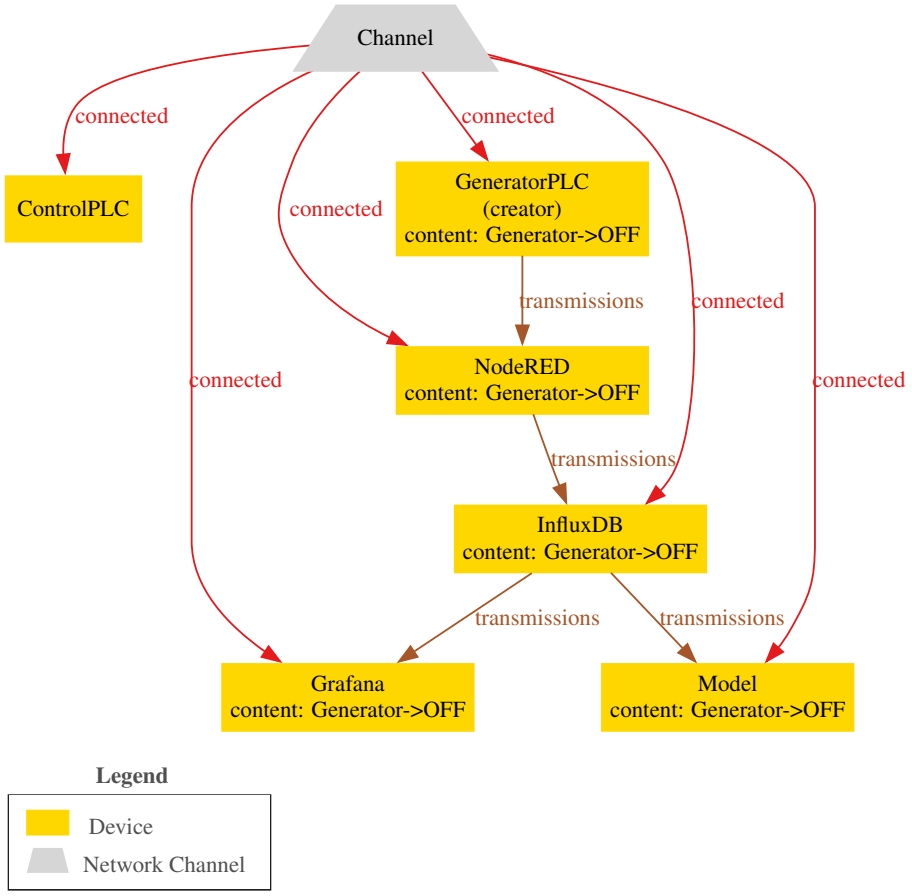


Figure 6.29: Demonstration of the new data modelling approach. Each device along the chain of transmissions has associated Tag-Value pairs, stored in the content relation. Multiple tags can be shared in the same dataset. While all devices here are Clean, a compromised device can modify the data that is shared. Each transmission has an associated Message exchange, for clarity, these have been omitted.

```
lone var sig Threat{
   var controls: set Device,          // Devices controlled
   var recon: Device → (Port+IP),     // Known device data
   var users: set User,               // Known usernames
   var pwds: set Password,            // Known passwords
   var current_actions: some Action, // Actions being taken
   var actions: set Action,           // Previous actions
}{
   // Stores all executed actions
   all action: actions | once action in current_actions

   // Store data gathered through recon actions
   recon' = recon + actions.learn

   // Store users and passwords found on controlled machines
   users' → pwds' = controls.stored_logins

   // Only compromised devices can be controlled
   controls.status = Compromised

   // Compromised devices can use any known address or port
   all controlled: controls {
      controlled.address in (
         controlled.assigned_ip + recon[Device])
      controlled.port in recon[Device]
   }

   controls in controls'

   all device: controls {
      // Controlled devices were controlled originally
      (historically device in controls) or {
         // Or some action has brought them under control
         some action: LateralMovement & actions {
            device = action.target
}  }  }  }
```

Figure 6.30: The updated Data signature used to represent the data exchange across the digital thread. The mapping relation associates a Tag-Value pair with each device that interacts with data. The transmissions relation tracks the message exchanges that include this data.

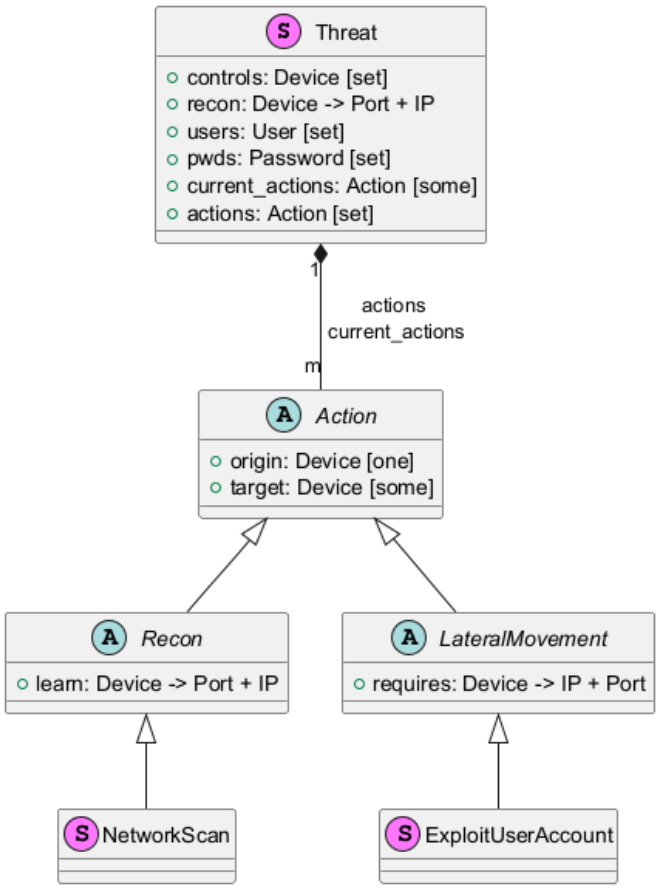


Figure 6.31: UML diagram of the action-based threat model used in our approach. A threat has a set of devices that it controls, and recon data that it has learned about the IP addresses and ports of the network's devices along with a set of known usernames and password. This data is then exploited through the execution of Actions. Recon actions gather data, LateralMovement actions exploit that data to gain control of additional devices.

Within our network scan we also represent an additional attack technique called port scanning. During a port scan, the attacker pings the ports of an identified IP address to identify any open ports that can be targeted for further actions. An attacker requires an IP address to know where in the network a device is located, but it also needs an open port through which it can interact with and attack the identified device. While these are two separate actions, in practice, they are executed together, so we have combined them to reduce the size of our model.

Once an attacker has gathered information about the network, they can begin to exploit it through `LateralMovement` actions. Lateral movement actions have requirements before they can be executed. In our case, they require the IP address of the device to be known and an open port identified. Using this information, the attacker can directly interact with the targetted system; in our system, this would allow them to bring up the login screen for a number of the components. If an attacker knows the correct username and password combination, they can exploit them to log into the identified system and alter its configuration and behaviour. The `ExploitUserAccount` action has the additional constraint that the appropriate user and

password combination must be known before the action can be executed. This data is gathered from other controlled devices.

6.8 Vulnerability Analysis

Using predicates, we can specify the type of behaviours that we want to search for. We can begin by confirming that our current specification displays normal operating behaviour without interference from an attacker. To do this we rely on a specification of normal operational behaviours, stating that uncompromised devices always share the information that they learn about the Process signature that is being monitored. This sharing of known data is how we represent the normal programming of the different devices. As such each type of device is configured to share that data with the appropriate device, e.g. PLCs share data with NodeRED, NodeRED shares data with InfluxDB. An instance showing the model's behaviour in this configuration is shown in Fig. 6.32. To check that the system is specified correctly, we remove the attacker from the modelled scenario and ask for any example where the instance of the model signature receives state data that does not match the state of the process signature in the scenario. Our search returns no counterexamples, allowing us to proceed with our analysis knowing that our designed system functions as intended. With this confirmation, we can introduce our threat model and search for any deviations in this behaviour to identify the impacts of our attacker.

We now introduce the attacker into the model. We allow the attacker access to a single device and the information contained within it to begin their attack. In our scenario, the attacker starts with control of an employee Workstation connected to the network but plays no role in the regular operation of the DT. We do not, at this stage, specify any structure to our network, allowing the topology to be a flat fully connected architecture. Asking alloy to search for examples of attacks reveals many weaknesses in this design. The first attack identified by our system is shown in Fig. 6.33. The flat network design allows for easy access to the vulnerable PLC devices of the system. No authentication is required for the attacker to communicate with them. All the attacker requires to interact with them is their IP address, which it gains through a network scan due to the flat structure of the network. The scenario shown here exemplifies the type of attacks tested for in Chapter 5 where an attacker modifies the state of the PLCs to undermine the system. In this case, the attacker has modified a Generator tag in the control PLC, such as Gen_A_Status or Gen_B_Status to be false, when the associated generator is active.

This attack demonstrates the need for network segmentation to protect the security of our system. In our next example, we isolate the PLCs, creating a separate channel for them, isolating them from most of the historian devices, except NodeRED and the workstation.

We introduce an instance of the Firewall subsignature of Router. We configure this and the layout of the network channels within it using the predicate shown in Fig. 6.34. We define the channels connected to it and their configuration, defining that one of them contains all of the

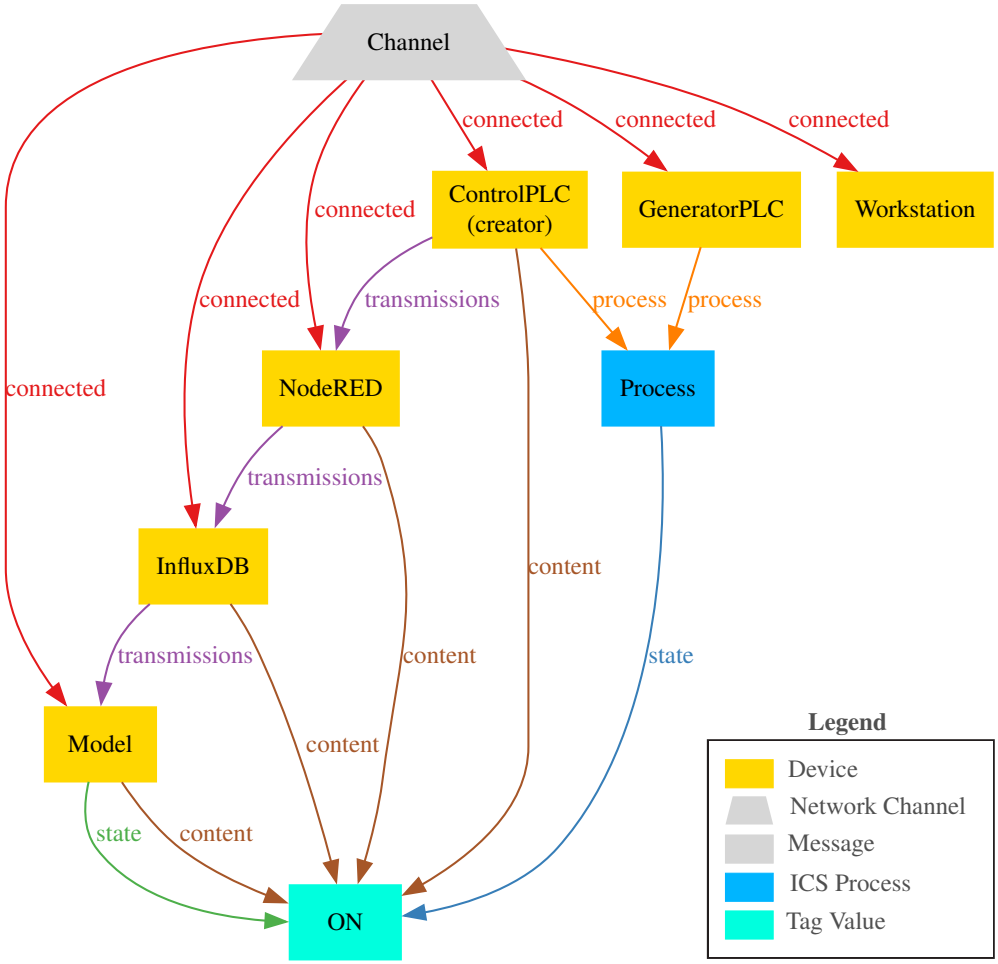


Figure 6.32: Demonstration of the specified system functioning without interference from an attacker. The state is projected to only show the tag values associated with the Sump_Pump at the process through the different devices in the Model. Since the values in the Process and Model are consistent for this Tag, and all the others, the system is performing as normal.

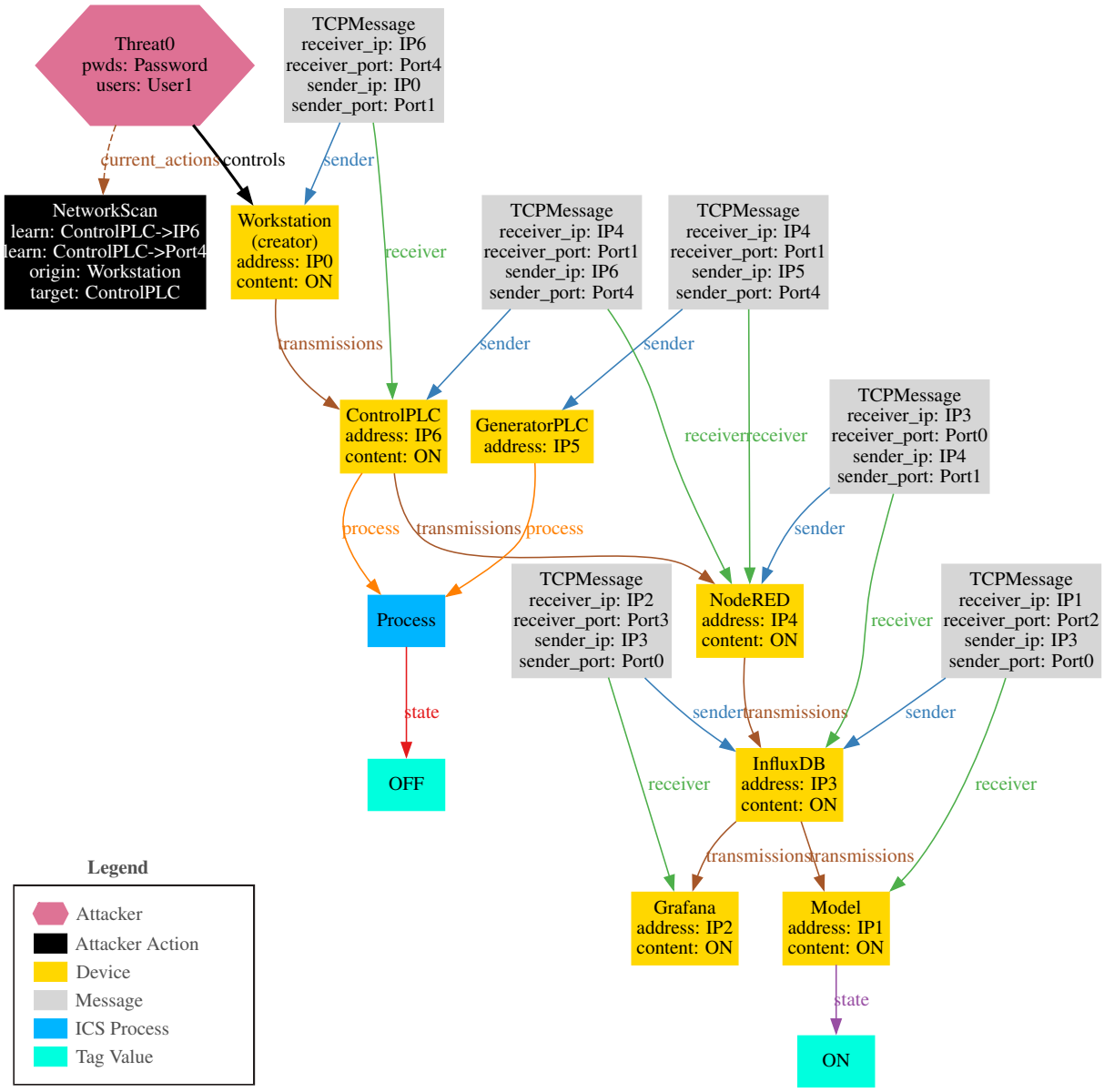


Figure 6.33: The first attack identified by our method. The attacker performs a network scan and detects the PLCs. Due to the weakness of the S7Comms protocols, the Control PLC accepts input from the attacker device, modifying its stored Generator variable. This false information is then passed through the historian devices to the DT model, which received an incorrect state.

```
pred Segmentation {
    #Router = 1
    one fw: Firewall {
        some disj chan1, chan2: Channel {
            fw.channels = chan1 + chan2
            PLC = chan1.connected
            fw.block =
              Workstation.assigned_ip → PLC.assigned_ip
}   }   }
```

Figure 6.34: An alloy predicate showing how a firewall can be configured in our model. The firewall connects a channel of PLCs to the rest of the network. It blocks all incoming PLC connections from the Workstation's assigned IP addresses.

PLCs. We then specify that the Firewall has a block relation between the assigned IP address of the workstation and those of the PLCs. Assuming that the workstation is not part of the typical workflow of the operators, there is no need for these devices to communicate, so they can be safely prevented from doing so.

Testing this new configuration reveals that this new network configuration is still insufficient to ensure the secure connection of the PLCs to the DT model. While the PLCs are now separated from the attacker, the remaining systems are still vulnerable, as Fig. 6.35 demonstrates. In this example, the attacker can gain sufficient information from a network scan to be able to interfere with the model collecting data from the Influx database. Since the threat actor gathers the IP addresses and ports of the two devices, it can observe their data exchange and attempt to inject false data into this process represented via a TCP message with the spoofed IP address of the Influx database. TCP messaging involves a three-way handshake that establishes a connection between two devices. While this exchange allows for the creation of a secure connection, it can be undermined through the deployment of MITM attacks, where an attacker intercepts the exchange and inserts themselves as an intermediary messenger. This grants the attacker the ability to intercept and modify all communication within the connection. An MITM can be performed by broadcasting fake Address Resolution Protocol (ARP) packets to devices in the network, causing them to send IP packets intended for another device to the attacking device instead. The generated example suggests that since the attacker can access the channel on which the devices are communicating and has gathered the required IP addresses of these two devices, it has sufficient resources to carry out this type of MITM attack.

To mitigate this type of attack, we must enhance the degree of segmentation within the network. While continuing to separate the PLCs from the historian devices, we should also separate the historian devices. Due to their integration with the DT model, they are even more critical to the operation of the dam than before and should be separated from other devices to protect them. We, therefore, enhance the Segmentation predicate shown previously to specify an

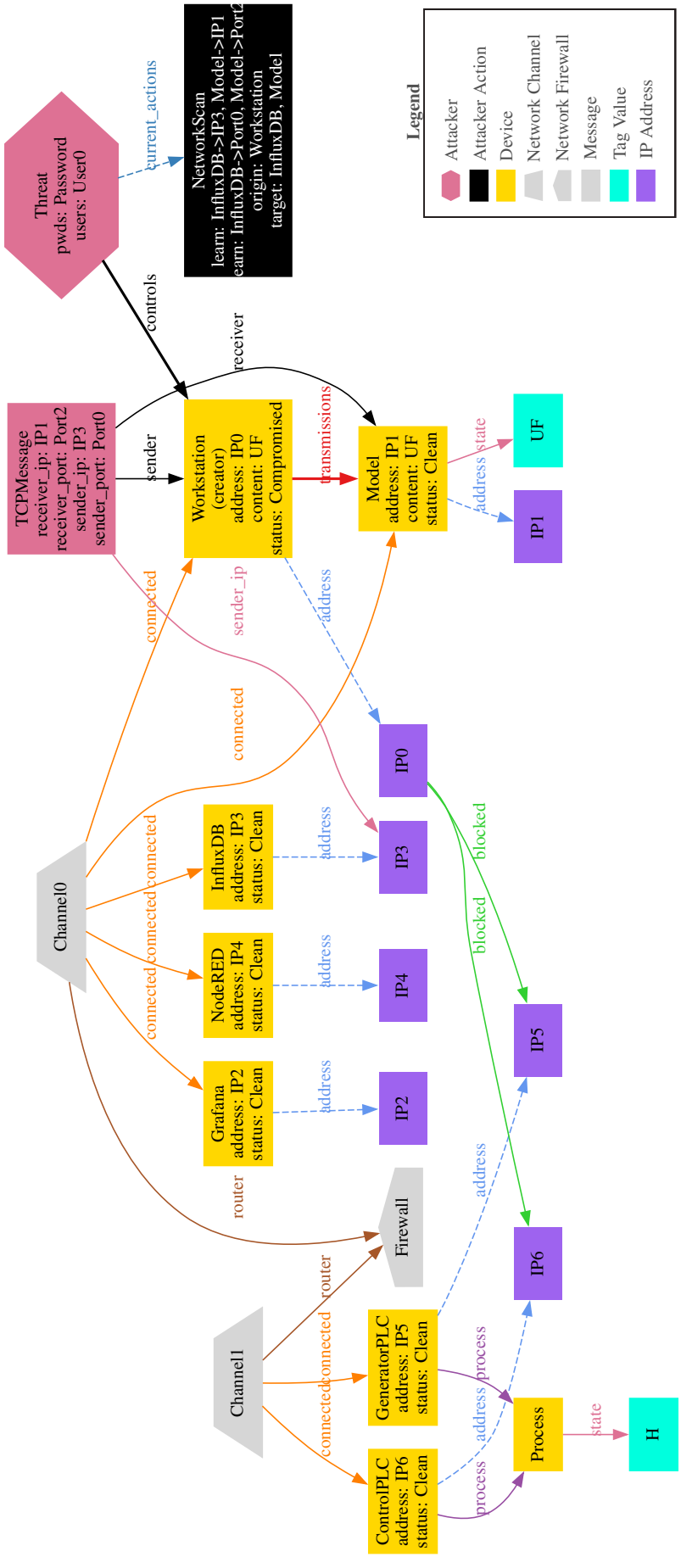


Figure 6.35: A man-in-the-middle attack that overcomes the introduction of a Firewall segmenting the network into PLCs and Historian devices. The threat scans the Model and the InfluxDB, learning their network configurations. By using this gathered data, the attacker has the ability to intercept, modify and inject data in a TCP message using a spoofed IP address to provide the Model with an incorrect Lower_Tank reading of Underfill (UF) when the correct reading is High (H). For readability the "blocked" relation has been projected as a binary relation across IP addresses, where it would usually display as a ternary relation originating from the Firewall.

additional channel. This new channel contains the NodeRED, InfluxDB and Grafana systems. It is connected to the PLC channel through a firewall as before to allow those systems to acquire the necessary operational data. These systems also need to be connected to the Model, which might suggest that the model should be positioned within this network, too. However, while in every example shown, the workstation has been used to undermine the system's operation, under normal circumstances, the workstation will be used by an employee who may require legitimate access to the insights and data of the DT model. Finding secure ways of enabling the model to be connected to the dam (via the historians) while also being accessible to employees outside the OT network is a key challenge of combining OT systems with IT infrastructure. Being able to visualise methods of addressing this challenge is one of the benefits of our approach.

We consider the deployment of a DMZ as a potential solution. We specify in our predicate that the model exists in a separate DMZ channel containing only the model. The devices within the DMZ can connect to the historian components via a firewall. The DMZ is also accessible to network devices outside the Workstation through an outwardly facing firewall. In this new configuration, both the PLCs and historian devices are insulated from the enterprise network with the employee workstations. This separation between the different types of devices implements the Purdue model, creating obstacles for attackers who attempt to penetrate the network from outside the firewalls.

Using this new DMZ specification, we can check the model for instances where the state of the Process and the state DT model do not align, as before. Our changes prevent the threat actor from observing data transmission to the model since the Workstation is outside of the channels where the exchange is occurring. To gain access to the network, the attacker will need to infiltrate the DMZ.

Checking the model gives the instance shown in Fig 6.36 showing how the attacker can gain access to the model. The threat actor begins by executing a network scan of the DMZ to gather the IP address and port number of the model located within it. With this recon knowledge, the attacker can execute a LateralMovement action, allowing the attacker to move through the network, expanding its influence. To perform a LateralMovement action the threat actor must have the IP address and port number of the target device. In our model we include the ExploitUserAccount signature implementing the LateralMovement abstract signature. This signature has the additional requirement that before it can be executed, the attacker must have access to the username and password for the target device. In the generated attack, the user credentials for the DT model have been stored on the Workstation, allowing the attacker to acquire them. Once the attacker has access to the DT model, it is able to modify its state as it wishes, either by modifying the data within it or by changing the model's behaviour.

The attack against our DMZ configuration shows the importance of password management in securing an ICS and ensuring an effective segmentation of OT and IT networks when a DMZ is deployed. We can test the effectiveness of employing good password management practices,

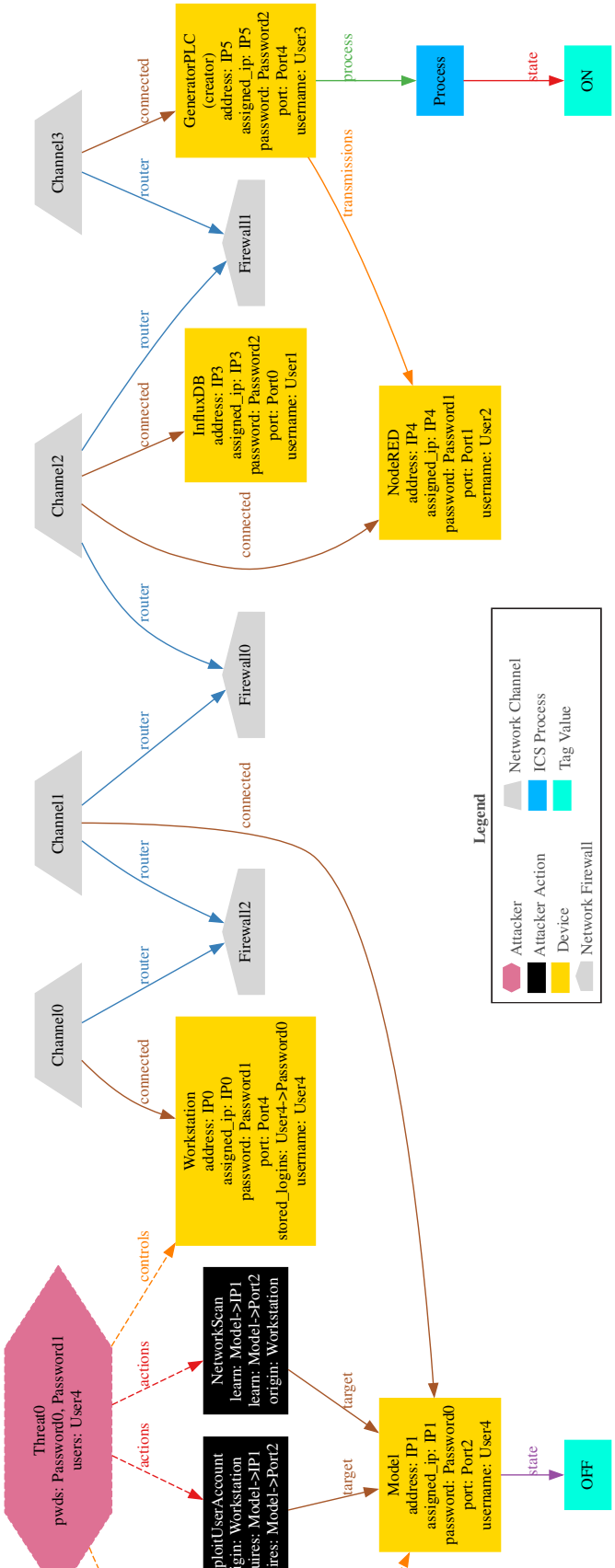


Figure 6.36: A vulnerability of the DMZ network structure is identified. The Workstation has a set of login credentials for the model stored within it. The attacker can scan the DMZ for the IP address and port of the Model and then use these acquired credentials to gain access to the Model and change its state. For clarity, some of the devices have been omitted from the visualised instance.

Device	Channel	Action	Steps	Time (ms)
7	3	5	5	85837
10	5	5	5	94540
15	7	5	5	403614
10	5	5	5	179257
10	5	10	5	188646
10	5	15	5	213234
10	5	10	1	9075
10	5	10	5	193973
10	5	10	10	938017
15	10	10	10	4539060

Table 6.3: Execution runtimes for Alloy to search for attacks under different scope size. In each case, the scope for unspecified signatures is 10. All runtimes are the median of 5 executions.

constraining that no usernames and passwords are stored on any device. The specification of this can readily be added to the predicate where we specified the creation of the DMZ. Running the model with this new constraint returns no counterexamples. There is no method by which the Threat can interfere with the normal transmission of data between the process and its DT model. Therefore, the DT model and its process have the same state in every variation.

6.9 Experimental Results

We use a similar approach as that used in Section 6.6 to evaluate the performance of this expanded model. We removed the DMZ from the model described at the end of Section 6.8 to increase the attack surface. To evaluate the system on an increased network size, we removed the abstract stipulation on the Device signature to allow additional devices to be added to the model. We then search for examples where the attacker can take control of the entire network. This should be impossible with the password management requirement still in effect however it forces the Glucose SAT solver to search the whole scope for satisfying instances. The results of this evaluation are presented in Table 6.3.

As can be seen from the results, Alloy can analyse the model for small network sizes within a relatively short period. The scopes shown in the table demonstrate the expected execution times for normal to extreme scope sizes for our model. Larger network sizes increase the time taken to search the scope, as expected. However, the effect is less pronounced than that of increasing the number of steps analysed. This indicates that the approach can scale to a larger network size of approximately double that which we demonstrated. At large step counts of 10, the long traces slow the analysis considerably. However, since multiple actions can be executed during each step, these long traces are rarely necessary to identify the presence of an attack, meaning the analysis terminates early. None of the attacks shown in Section 6.8 required more than three

steps to analyse. Most promisingly, the impact of increasing the number of actions within the model was relatively minimal by comparison. This suggests that the complexity of the attacker model can be increased without dramatically increasing the time taken to analyse the model.

6.10 Discussion

The elements in our model have been used to demonstrate the utility of Alloy and our modelling framework as a means of constructing cyber security-oriented representations of a system's design. However, Alloy's declarative language means that all of the signatures in the developed models can be extended further to suit specific use cases. We have demonstrated this capability in three ways.

The components within our model serve as an example of how the device signature can be expanded during its application to a real system. Abstract signatures can be created to facilitate this process; for example, our PLC signature can be extended to represent other PLCs in similar systems, requiring only an additional signature with assignments to the appropriate fields.

The threat model can be expanded to represent different types of attack techniques depending on the network and its components. We've provided the Recon and LateralMovement action signatures to show how different types of action can be grouped together. Further Recon and LateralMovement actions can be created to target specific types of device, or non-device signatures such as messages or routers. Additional mitigation strategies can be integrated into the model accordingly and Alloy can help identify how best to integrate them by identifying means by which they can be undermined.

Finally, how messages and data are exchanged can be expanded. Our model represents TCP and IP messages and uses a single Data signature as a payload. However, alternative implementations of the Message signatures can be created to represent other messaging protocols carrying different types of data. Of particular note, the creation of an encrypted message type could be used, in combination with additional attack actions, to check for attacks involving message interceptions and tampering. Alloy can also specify sequences of events, allowing message exchange processes to be represented.

6.11 Summary

We have demonstrated how Alloy models can be developed to construct representations of ICS networks containing DTs. We have shown different approaches to represent devices communicating within a network, the channels they communicate across and the movement of data within those channels. We demonstrate how to apply this approach through an initial example of a non-specific DT within an ICS environment showing how the Alloy Analyzer can simulate threats to the network through its ability to generate counterexamples with attacker nodes. We constructed

properties for two variations of spoofing attacks: ARP spoofing and IP spoofing. The model and properties were used to check for the presence of these attacks, resulting in traces that revealed vulnerabilities that undermined the performance of a DT.

We then refined these models to construct a representation specifically to analyse the ability of security threats to undermine the performance of a DT. We created a unique representation of data moving through the network between the DT and the process that it is twinned to. Our representation of Data utilised a ternary relational mapping of devices to tag value pairs. This mapping was used to model the movement of key datasets across the network to the DT. We then use our specification of messages to move this data between devices, creating relational mappings for each device that interacts with it. We demonstrated how, with some constraints about how devices move this data across the network, this data signature represents the digital thread in a DT.

To demonstrate the application of this method, we applied it to our own DT framework, developed in Chapters 4 and 5 and analysed it under different configurations. We constructed representations of each of the DT components of our hydroelectric dam and specified their arrangement, configuration and interactions to demonstrate how our Alloy model represents normal operation of a DT in an ICS.

A key feature of our model is the development of a novel approach to representing an attacker and their interactions with the network. We represent an attacker as a threat entity within the network that is able to combine the knowledge that they gain from their interactions within the system. They are able to use actions to learn data about the network and gain control of devices by exploiting gathered data. These actions are executed in sequence to create an attack trace of an attacker traversing a network and exploiting vulnerabilities.

We integrated our threat model into our representation of the hydroelectric DT framework and used it to identify vulnerabilities and attack patterns. Through an iterative refinement process, our model was used to deploy cyber defence mitigations to identify a more secure network architecture implementing the Purdue model. We also showed how even this more secure network architecture can be undermined by users using poor password management techniques.

We conclude by presenting performance data for the developed approach when analysed by the Alloy Analyzer. We found that the analysis concluded within an acceptable duration, even for scopes larger than the ones used to perform our own analysis, indicating the potential for the approach to handle networks containing more devices. Finally, we discussed the extensibility of our model to model these larger networks and expand the threat model. Our approach can be extended by enhancing data representation within the model to include dependencies, allowing for the downstream effects of compromised data reaching the DT to be represented. This supports the expansion of the components to model the interactions between the different layers which contextualised the impacts of attacker interactions upon DT infrastructure.

Chapter 7

Conclusions

The aim of this thesis has been to develop methods for securing DTs and the systems they mirror. This has been approached from two angles: (1) improving anomaly detection in DT data and (2) developing a formal modelling framework to analyse cybersecurity vulnerabilities in DT architectures. Our contributions are:

- The creation of a DT framework for a pumped-storage hydroelectric dam testbed. The framework implements a historian to pull data from two PLCs that control the dam's generators and water feeds. The data within the historian is then used by the DT environment to assemble the data from the two PLCs into a single unified state that can be presented to DT models.
- An ICS anomaly detection method using a specification-based representation of PLC logic, modelled in Promela. We designed models to identify state data that indicated a deviation from normal operation. We integrated the SPIN model checker into the ICS environment to verify live operational data. The method achieved an F1-score of 99.52% and an accuracy of 99.05% when evaluated against a dataset of 558,140 test transitions.
- A series of Alloy models that enable security analysis and the identification of vulnerabilities in DT networks. The models uniquely represent data movement and cyber attacker actions, supporting analysis of the security of a DT design. The benefit of this approach has been demonstrated through its application to the hydroelectric dam framework developed in this thesis, leading to a refined network architecture that mitigates security risks.

7.1 Overview

Taken together, the contributions within this thesis present methods of developing more secure DTs that ensure the resilience of the systems that they monitor. DTs can be used to provide many benefits to the system to which they are connected. However, if they are not developed with security as a primary concern, they may undermine the reliability of the whole ICS.

In Chapter 4, we created a DT framework upon which to base our formal methods approaches. The developed infrastructure connects OT components to digital models through the use of NodeRED flows, storing data in a time-series InfluxDB database. The data can be visualised for operators using Grafana or presented to digital models through a developed DT module that collects and synchronises PLC-generated state data. The approach was able to track system states, however, establishing a system baseline of normal behaviours was challenging. This inspired the search for a means to identify anomalous system data received by the DT.

In Chapter 5, we develop Promela models from the PLC Ladder Logic to specify normal system behaviour without requiring comprehensive system test data upon which to construct a baseline. We document the transcription process, showing how combining `goto` and `d_step` statements can replicate the control flow of ladder logic rungs and the simultaneous writing of system outputs. The Promela models can predict concurrent execution of the PLCs, defining the set of all reachable states by interleaving individual PLC behaviours.

We developed two Promela models to examine the received PLC states; a *trunk* model and a *branch* model. The trunk model was shown to correctly identify all of the normal baseline behaviours in our test set while only incorrectly recognising a small number of anomalous states. This model achieved very high accuracy and F1-scores of 99.96% and 99.98%, respectively. We developed a second model (the branch model) to identify if the observed transitions between consecutive sets of gathered state data were valid. Though the branch model performed worse when tested on anomalous transitions originating from invalid states, it performed well on transitions that originated from valid states, demonstrating a strong ability to determine if the transition resulted in a valid state. The branch model achieved a lower accuracy and F1-score of 72.66% and 84.16%. An overall F1-score of 99.52 calculated on the results of an evaluation on 558,140 state transitions determined that the two models complement each other well and, when used together, demonstrate a strong ability to detect invalid system data and transitions. We explain methods for integrating SPIN into the ICS environment to perform analysis on real-time data as it is collected from the PLCs. The inclusion of this approach enhances the reliability of DT models by enabling the identification of system behaviour anomalies before they are processed. This allows operators to detect potential threats early and swiftly initiate appropriate cyber response actions.

In Chapter 6, we used Alloy to support the secure integration of DT into an ICS system. We introduce methods of representing DT and ICS networks and show how these can be used to identify IP and ARP spoofing attacks. We then proceed to refine an approach, presenting a method that directly targets the challenge of developing secure ICS network architectures that incorporate DTs. By using a refined Data signature, the developed models could represent normal data flow within a DT network. We showed how our refined approach contextualises the impacts of vulnerabilities in the digital thread in real terms by representing the hydroelectric dam testbed DT developed in Chapters 4 and 5.

We developed a novel threat model that uses an extensible framework of actions comprised of real attack techniques to infiltrate our network. The developed actions allow the attacker to gather data and utilise it to expand their influence by exploiting devices. Through the specification of properties, we are able to identify specific sequences of actions that the threat actor can take to undermine the critical operation of the DT. We use the attack traces generated by Alloy to identify vulnerabilities in our system and refine defensive mitigation strategies. Through this analysis, we were able to develop an improved architecture using network segmentation to implement a DMZ. Analysis of the improved architecture demonstrates that it promotes the robust operation of the DT when combined with secure password management from users.

Our Alloy modelling approach enables system designers to develop secure network configurations for DT deployment. The flexibility of the Alloy specification language enables our models to be extended and applied to systems with different devices and configurations. Our threat model can grow and evolve to analyse new attack techniques as they emerge. Even after deployment, our network models help communicate threats to operators, providing a clearer understanding of the evolving threat landscape and enhancing cybersecurity awareness among users.

7.2 Limitations

The approaches developed in this thesis show promising results but also have certain limitations. The effectiveness of model checking approaches is derived from the ability of the model checker to search the model's state space comprehensively. However, in both of the used methods, as model size and scope increase, analysis times grow exponentially. Our results demonstrate that this does not hinder the practical utility of our methods for the tested applications. In this section, we discuss the potential impact of this limitation on the scalability of our approaches.

7.2.1 SPIN

Firstly, since our approach depends upon model checking, the state-space explosion problem limits the size of the Promela model that can be analysed. This means that the approach cannot be directly applied to large-scale systems without some level of abstraction. In our example, we required minimal abstraction of system behaviours except in the case of analogue system variables, such as timers, which had to be discretised. This limitation of our approach means that it is best suited to a small number of PLC programs utilising discrete variables that work in close coordination.

Another challenge of applying our SPIN modelling approach relates to the integration of the model checker into our framework to operate on real-time historian data. SPIN processes data slower than it is received, taking on average 18.2 seconds to execute the models on data that is

collected every 0.8 seconds. Multi-threading and eliminating the rechecking of the previously observed system state can reduce this, however the best mitigation is the integration of a memoisation approach. By storing and retrieving previously computed results, the system stabilised after an initial delay caused by the validation of baseline behaviours. Once stabilised, anomaly detection became more efficient and was able to prioritise newly observed data rapidly, with anomalies being identified within a few seconds.

Finally, our approach requires transcription of the PLC code to Promela, which currently requires details knowledge of model checking and the Promela language. To assist this process, we describe how we performed the transcription process. We believe that the use of `d_step` commands in combination with `jump` statements can be readily applied to model many of the PLC Ladder Logic components. However, this still presents an obstacle towards the implementation of our approach that would be best addressed through a partial or fully automatic transcription method.

7.2.2 Alloy

The state-space explosion problem is also a challenge for our Alloy modelling approach. We've demonstrated how pushing the analysis to search for large attack traces on networks of 15 or more devices can cause the analysis to take up to an hour. While this can be reduced by decreasing the scope of other components within the model and utilising higher-powered computing solutions, a hard limit of feasible analysis exists. Since ICS can consist of a large number of connected devices, abstraction of system components or a focus on specific subsystems is required to apply our models to large-scale industrial systems. We believe our approach is still beneficial with this abstraction since it is not necessary to represent every possible end controller in the system to address key system vulnerabilities. If the security of each type of device can be established, that analysis can be extended to other devices that are isometrically equivalent. Our analysis, therefore, only needs to include those devices that present unique characteristics or which are configured in unique ways.

A further consideration of our approach is derived from the bounded analysis of Alloy. Due to the undecidability of first-order logic that Alloy uses, a bounded scope is provided to enforce that its analysis will terminate. The analysis is therefore only correct within the specified scope of analysis. The implication of this for our analysis is that the approach cannot guarantee that an attack trace will not be found at a larger scope, i.e. if an attacker had unlimited actions, they may eventually be able to infiltrate the system. However, the small scope hypothesis [115] suggests that, for most problems, if a counterexample demonstrating a viable attack sequence exists, it can be found at a relatively small scope. In our testing, this was observed to be the case with all the identified attacks occurring well within the bounds of analysis and being identified quickly. Furthermore, our evaluation of the execution times at varying numbers of actions suggests that increasing the number of actions has a relatively small impact on analysis times. Therefore,

even at the upper limits of network size, simulating attacks with a large enough number of attack actions should not be an obstacle to analysis.

7.3 Future Work

We conclude by considering some avenues for further work to expand and improve the applicability of our methods.

The validation of PLC programs through model checking is an area of research that is related to our specification-based anomaly detection approach. Tools such as PLCverif [52, 215] allow for the automatic construction of representations of PLC programs into formats that can be analysed using model checkers for safety and reliability, such as in [3]. However, PLCverif could not be used to represent Ladder Logic in Promela as it currently only supports the conversion of Statement List (STL) PLC programs into specifications for NuSMV, nuXmv, Theta, and CBMC [215]. Nonetheless, we have demonstrated that SPIN can be effectively used to monitor normal ICS operation by identifying valid system-wide states arising from the combined execution of individual components. While the application of our approach currently requires the manual transcription process of converting PLC programs to Promela for analysis, we have explained the method use to achieve this for the PLC logic contained within the hydroelectric dam. Future work could examine the extensibility of this to other PLC programming constructs and examine methods of automating some or all of this process.

Analysing a more extensive ICS system with SPIN could cause issues with scalability as the number of possible states increases. Further work could consider layering the implementations of our IDS approach to perform anomaly detection at different levels of supervisory systems. Subsystems could utilise detailed models of all the subsystem tag values, checking for anomalies at a PLC level. A site-wide supervisory model could then use abstraction to check the subprocess behaviours within the context of the overall system, ensuring that the subsystems are correctly coordinated.

As discussed in Section 6.10, our Alloy model can be expanded to model different types of ICS components, their configuration and the threats that they face. In addition to the possibility of expanding the Device, Message, Action and Router signatures, to aid this expansion, we identify some other areas of potential development in this type of Alloy modelling.

Our user model is a flat hierarchy of users who are all equally authorised for all hosts that they access. User privileges could be integrated into the model such that different usernames and passwords could have different levels of access to the system. A ternary relation would be well-suited to this process, mapping user and password combinations to an enumerated set of access rights within each device. The access rights can then be incorporated as a requirement for certain actions to be executed.

We have demonstrated how additional messages with different properties can be added. The

devices that can send and receive these different messages can be constrained accordingly. For example, the PLCs communicate with NodeRed via the unencrypted TCP-based S7 communication protocol. Message properties can be implemented to represent properties such as encryption or access tokens. This enables additional actions for the attacker since they can intercept or sniff the packets being exchanged on the network without requiring access to the PLCs or NodeRED. Access tokens can be modelled similarly to usernames and passwords.

The extensibility of our method means that it can be used to model threats and identify vulnerabilities beyond those we have demonstrated. Future work could expand the model to accommodate different system requirements, additional properties, and the continuously evolving cyber threat landscape. The features discussed in this section represent just a few potential directions for exploring the limits of this extensibility within the bounds of feasible computational analysis.

Bibliography

[1] *Static Analysis*, pages 1–36. Springer Netherlands, Dordrecht, 2008.

[2] IEA (2021). Key world energy statistics 2021. Accessed: 12/11/2024.

[3] Zsófia Ádám, Ignacio David López Miguel, Anastasia Mavridou, Thomas Pressburger, Marcin Bęś, Enrique Blanco Viñuela, Andreas Katis, Jean-Charles Tournier, Khanh V Trinh, and Borja Fernández Adiego. Automated verification of programmable logic controller programs against structured natural language requirements. 2023.

[4] Sridhar Adepu and Aditya Mathur. Using process invariants to detect cyber attacks on a water treatment system. In Jaap-Henk Hoepman and Stefan Katzenbeisser, editors, *ICT Systems Security and Privacy Protection*, pages 91–104, Cham, 2016. Springer International Publishing.

[5] Borja Fernández Adiego, Dániel Darvas, Jean-Charles Tournier, Enrique Blanco Viñuela, and Víctor M. González Suárez. Bringing automated model checking to plc program development — a cern case study —. *IFAC Proceedings Volumes*, 47(2):394–399, 2014. 12th IFAC International Workshop on Discrete Event Systems (2014).

[6] Siemens AG. S7 communication with PUT/GET. Technical report, Siemens AG, 2020.

[7] Akers. Binary decision diagrams. *IEEE Transactions on computers*, 100(6):509–516, 1978.

[8] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, 17(4):2347–2376, 2015.

[9] Aliaa M Alabdali, Lilia Georgieva, and Greg Michaelson. Modelling of secure data transmission over a multichannel wireless network in alloy. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 785–792. IEEE, 2012.

[10] Cristina Alcaraz and Javier Lopez. Digital twin: A comprehensive survey of security threats. *IEEE Communications Surveys & Tutorials*, 24(3):1475–1503, 2022.

[11] Tejasvi Alladi, Vinay Chamola, and Sherali Zeadally. Industrial control systems: Cyberattack trends and countermeasures. *Computer Communications*, 155:1–8, 2020.

[12] José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto, and Simão Melo de Sousa. An overview of formal methods tools and techniques. *Rigorous Software Development: An Introduction to Program Verification*, pages 15–44, 2011.

[13] Wael Alsabbagh and Peter Langendörfer. Security of programmable logic controllers and related systems: Today and tomorrow. *IEEE Open Journal of the Industrial Electronics Society*, 4:659–693, 2023.

[14] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for real-time systems. In *[1990] Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 414–425. IEEE, 1990.

[15] E Archana, Akshay Rajeev, Aby Kuruvila, Revathi Narayankutty, and Jinesh M Kannimoola. A formal modeling approach for qos in mqtt protocol. In *Data Communication and Networks*, pages 39–57. Springer, 2020.

[16] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. In *Twenty-first international joint conference on artificial intelligence*. Citeseer, 2009.

[17] David Bailey and Edwin Wright. *Practical SCADA for industry*. Elsevier, 2003.

[18] Davide Basile, Alessandro Fantechi, and Irene Rosadi. Formal analysis of the unisig safety application intermediate sub-layer. In Alberto Lluch Lafuente and Anastasia Mavridou, editors, *Formal Methods for Industrial Critical Systems*, pages 174–190, Cham, 2021. Springer International Publishing.

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

[20] Fabian Becker, Pascal Bibow, Manuela Dalibor, Aymen Gannouni, Viviane Hahn, Christian Hopmann, Matthias Jarke, Istvan Koren, Moritz Kröger, Johannes Lipp, et al. A conceptual model for digital shadows in industry and its application. In *Conceptual Modeling: 40th International Conference, ER 2021, Virtual Event, October 18–21, 2021, Proceedings 40*, pages 271–281. Springer, 2021.

[21] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *QEST*, volume 6, pages 125–126. Citeseer, 2006.

[22] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and software verification: model-checking techniques and tools*. Springer Science & Business Media, 2013.

[23] Robin Berthier and William H. Sanders. Specification-based intrusion detection for advanced metering infrastructures. page 184 – 193, 2011.

[24] Yves Bertot, Temur Kutsia, and Michael Norrish, editors. *LIPIcs, Volume 309, ITP 2024, Complete Volume*, volume 309 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[25] Dirk Beyer and Thomas Lemberger. Software verification: Testing vs. model checking: A comparative evaluation of the state of the art. In *Hardware and Software: Verification and Testing: 13th International Haifa Verification Conference, HVC 2017, Haifa, Israel, November 13-15, 2017, Proceedings 13*, pages 99–114. Springer, 2017.

[26] Christopher Bissell. A history of automatic control. *Springer handbook of automation*, pages 53–69, 2009.

[27] Ron Bitton, Tomer Gluck, Orly Stan, Masaki Inokuchi, Yoshinobu Ohta, Yoshiyuki Yamada, Tomohiko Yagyu, Yuval Elovici, and Asaf Shabtai. Deriving a cost-effective digital twin of an ics to facilitate security evaluation. In *Computer Security: 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I 23*, pages 533–554. Springer, 2018.

[28] Dines Bjørner. The vienna development method (vdm) software specification & program synthesis. In *Mathematical Studies of Information Processing: Proceedings of the International Conference Kyoto, Japan, August 23–26, 1978*, pages 326–359. Springer, 2005.

[29] Dines Bjørner and Cliff B Jones. *The Vienna development method: The meta-language*. Springer, 1978.

[30] Scott Steele Buchanan. Cyber-attacks to industrial control systems since stuxnet: A systematic review. 2022.

[31] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, February 1990.

[32] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, pages 334–342, Cham, 2014. Springer International Publishing.

[33] Advanced Manufacturing Research Centre. Untangling the requirements of a digital twin. Technical report, The University of Sheffield, October 2020.

[34] Himadri Chauhan, Vipin Kumar, Sumit Pundir, and Emmanuel S Pilli. A comparative study of classification techniques for intrusion detection. In *2013 International Symposium on Computational and Business Intelligence*, pages 40–43. IEEE, 2013.

[35] Shengbo Chen, Hao Fu, and Huaikou Miao. Formal verification of security protocols using SPIN. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, pages 1–6, 2016.

[36] Samir Chouali, Azzedine Boukerche, and Ahmed Mostefaoui. Towards a formal analysis of mqtt protocol in the context of communicating vehicles. In *Proceedings of the 15th ACM International Symposium on Mobility Management and Wireless Access*, pages 129–136, 2017.

[37] Paul Cichonski, Tom Millar, Tim Grance, and Karen Scarfone. Computer security incident handling guide. Technical report, National Institute for Standards and Technology, 2024.

[38] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. Nusmv: A new symbolic model verifier. In Nicolas Halbwachs and Doron Peled, editors, *Computer Aided Verification*, pages 495–499, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[39] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An open-source tool for symbolic model checking. In *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings 14*, pages 359–364. Springer, 2002.

[40] Edmund M Clarke. Model checking. In *Foundations of Software Technology and Theoretical Computer Science: 17th Conference Kharagpur, India, December 18–20, 1997 Proceedings 17*, pages 54–56. Springer, 1997.

[41] Edmund M. Clarke and Somesh Jha. Symmetry and induction in model checking. In Jan van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, volume 1000, pages 455–470, 1995.

[42] Edmund M. Clarke, Somesh Jha, Reinhard Enders, and Thomas Filkorn. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9:77–104, 1996.

[43] Edmund M. Clarke and Jeannette M Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643, 1996.

[44] Edmund M. Clarke and Jeannette M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643, 1996.

[45] Kate Conger. Driver charged in Uber's fatal 2018 autonomous car crash. *The New York Times*, 2020. Retrieved from `https://www.nytimes.com/2020/09/15/technology/uber-autonomous-crash-driver-charged.html`.

[46] Mauro Conti, Denis Donadel, and Federico Turrin. A survey on industrial control system testbeds and datasets for security research. *IEEE Communications Surveys & Tutorials*, 23(4):2248–2294, 2021.

[47] David Cooper and Janet Bernes. Tokeneer ID Station EAL5 Demonstrator: Summary Report. Technical report, Praxis, 2008. Accessed: 16/01/2025.

[48] MITRE Corporation. MITRE ATTCK framework. `https://attack.mitre.org/`. Accessed: 16/01/2025.

[49] MITRE Corporation. MITRE ATTCK framework. `https://attack.mitre.org/matrices/ics/`. Accessed: 10/12/2024.

[50] MITRE Corporation. MITRE ATTCK framework. `https://attack.mitre.org/matrices/enterprise/`. Accessed: 10/12/2024.

[51] Anthony Craig and Brandon Valeriano. Conceptualising cyber arms races. In *2016 8th international conference on cyber conflict (CyCon)*, pages 141–158. IEEE, 2016.

[52] Dániel Darvas, Enrique Blanco Vinuela, and Borja Fernández Adiego. PLCverif: A Tool to Verify PLC Programs Based on Model Checking Techniques. page WEPGF092, 2015.

[53] Dániel Darvas, István Majzik, and Enrique Blanco Viñuela. Plc program translation for verification purposes. *Periodica Polytechnica Electrical Engineering and Computer Science*, 61(2):151–165, 2017.

[54] Shoumen Datta. Digital twins. 2017.

[55] Antonio João Gonçalves de Azambuja, Tim Giese, Klaus Schützer, Reiner Anderl, Benjamin Schleich, and Vilson Rosa Almeida. Digital twins in industry 4.0 – opportunities and challenges related to cyber security. *Procedia CIRP*, 121:25–30, 2024. 11th CIRP Global Web Conference (CIRPe 2023).

[56] Michael Denzel, Mark Ryan, and Eike Ritter. A malware-tolerant, self-healing industrial control system framework. In *ICT Systems Security and Privacy Protection: 32nd IFIP TC 11 International Conference, SEC 2017, Rome, Italy, May 29-31, 2017, Proceedings 32*, pages 46–60. Springer, 2017.

[57] Petr N Devyanin, Alexey V Khoroshilov, Victor V Kuliamin, Alexander K Petrenko, and Ilya V Shchepetkov. Formal verification of os security model with alloy and event-b. In *Abstract State Machines, Alloy, B, TLA, VDM, and Z: 4th International Conference, ABZ 2014, Toulouse, France, June 2-6, 2014. Proceedings 4*, pages 309–313. Springer, 2014.

[58] Alessandro Di Pinto, Younes Dragoni, and Andrea Carcano. Triton: The first ics cyber attack on safety instrument systems. *Proc. Black Hat USA*, 2018:1–26, 2018.

[59] Henning Dierks. Synthesizing controllers from real-time specifications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(1):33–43, 1999.

[60] Marietheres Dietz and Gunther Pernul. Unleashing the digital twin's potential for ics security. *IEEE Security Privacy*, 18(4):20–27, 2020.

[61] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[62] JVS Do Amaral, CH Dos Santos, JAB Montevechi, and AR De Queiroz. Energy digital twin applications: a review. *Renewable and Sustainable Energy Reviews*, 188:113891, 2023.

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

[64] Mark Dowson. The ariane 5 software failure. *ACM SIGSOFT Software Engineering Notes*, 22(2):84, 1997.

[65] Matthias Eckhart and Andreas Ekelhart. Towards security-aware virtual environments for digital twins. pages 61–72, 05 2018.

[66] Matthias Eckhart, Andreas Ekelhart, David Allison, Magnus Almgren, Katharina Ceesay-Seitz, Helge Janicke, Simin Nadjm-Tehrani, Awais Rashid, and Mark Yampolskiy. Security-enhancing digital twins: Characteristics, indicators, and future perspectives. *IEEE Security Privacy*, 21(6):64–75, 2023.

[67] Maddy Ell and Saman Rizvi. Cyber security breaches survey 2024. Technical report, Department for Science, Innovation & Technology, 2024. Accessed: 16/01/2025.

[68] Tolga Erol, Arif Furkan Mendi, and Dilara Doğan. The digital twin revolution in healthcare. In *2020 4th international symposium on multidisciplinary studies and innovative technologies (ISMSIT)*, pages 1–7. IEEE, 2020.

[69] Mustafa Ersan and Erdal Irmak. Development and integration of a digital twin model for a real hydroelectric power plant. *Sensors*, 24(13), 2024.

[70] Stephen Ferguson. Apollo 13: The first digital twin. `https://blogs.sw.siemens.com/simcenter/apollo-13-the-first-digital-twin/`. Accessed: 16/01/2025.

[71] Borja Fernández Adiego, Dániel Darvas, Enrique Blanco Viñuela, Jean-Charles Tournier, Simon Bliudze, Jan Olaf Blech, and Víctor Manuel González Suárez. Applying model checking to industrial-sized plc programs. *IEEE Transactions on Industrial Informatics*, 11(6):1400–1410, 2015.

[72] Wan Fokkink and Jun Pang. Simplifying itai-rodeh leader election for anonymous rings. *Electronic Notes in Theoretical Computer Science*, 128(6):53–68, 2005.

[73] Christopher A. Ford. The trouble with cyber arms control. *The New Atlantis*, pages 52–67, 2010.

[74] Javier Franco, Ahmet Aris, Berk Canberk, and A Selcuk Uluagac. A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems. *IEEE Communications Surveys & Tutorials*, 23(4):2351–2383, 2021.

[75] Douglas Fraser, Ruben Giaquinta, Ruth Hoffmann, Murray Ireland, Alice Miller, and Gethin Norman. Collaborative models for autonomous systems controller synthesis. *Formal Aspects of Computing*, 32(2):157–186, 2020.

[76] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*, pages 379–395. Springer, 2011.

[77] Jonas Fritzsch, Tobias Schmid, and Stefan Wagner. Experiences from large-scale model checking: Verifying a vehicle control system with nusmv. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 372–382, 2021.

[78] Aidan Fuller, Zhong Fan, Charles Day, and Chris Barlow. Digital twin: Enabling technologies, challenges and open research. *IEEE Access*, 8:108952–108971, 2020.

[79] Andrew Futter. *What does cyber arms control look like?: Four principles for managing cyber risk*. European Leadership Network., 2020.

[80] Joseph Gardiner, Barnaby Craggs, Benjamin Green, and Awais Rashid. Oops i did it again: Further adventures in the land of ics security testbeds. In *Proceedings of the ACM Workshop on Cyber-Physical Systems Security & Privacy*, pages 75–86, 2019.

[81] Rob Gerth. Concise Promela reference. `https://spinroot.com/spin/Man/Quick.html`, 1997. Accessed: 08/01/2025.

[82] Chaouki Ghenai, Lama Alhaj Husein, Marwa Al Nahlawi, Abdul Kadir Hamid, and Maamar Bettayeb. Recent trends of digital twin technologies in the energy sector: A comprehensive review. *Sustainable Energy Technologies and Assessments*, 54:102837, 2022.

[83] Edward Glaessgen and David Stargel. The digital twin paradigm for future nasa and us air force vehicles. In *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA*, page 1818, 2012.

[84] Philippe Glass and Giovanna Di Marzo Serugendo. Coordination model and digital twins for managing energy consumption and production in a smart grid. *Energies*, 16(22), 2023.

[85] P.R. Gluck and G.J. Holzmann. Using spin model checking for flight software verification. In *Proceedings, IEEE Aerospace Conference*, volume 1, pages 1–1, 2002.

[86] Grafana Labs. Grafana OSS. `https://grafana.com/grafana/download/10.3.1`, Jan 2024. v10.3.1.

[87] Michael Grieves. Digital model, digital shadow and digital twin. `https://www.linkedin.com/pulse/digital-model-shadow-twin-michael-grieves/`. Accessed: 16/01/2025.

[88] Michael Grieves and John Vickers. Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In *Transdisciplinary perspectives on complex systems*, pages 85–113. Springer, 2017.

[89] Michael W. Grieves. Digital twins: past, present, and future. In *The digital twin*, pages 97–121. Springer, 2023.

[90] Muluken Hailesellasie and Syed Rafay Hasan. Intrusion detection in plc-based industrial control systems using formal verification approach in conjunction with graphs. *Journal of Hardware and Systems Security*, 2:1–14, 2018.

[91] Tim Harford. High-frequency trading and the $440m mistake. `https://www.bbc.co.uk/news/magazine-19214294`. Accessed: 09/01/2025.

[92] K. Havelund, M. Lowry, and J. Penix. Formal analysis of a space-craft controller using spin. *Software Engineering, IEEE Transactions on*, 27:749–765, 09 2001.

[93] Manu S Hegde, HK Jnanamurthy, and Sanjay Singh. Modelling and verification of extensible authentication protocol using spin model checker. *International Journal of Network Security & Its Applications*, 4(6):81, 2012.

[94] Thomas A Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and computation*, 111(2):193–244, 1994.

[95] Manuel V Hermenegildo and José F Morales. *Static Analysis: 30th International Symposium, SAS 2023, Cascais, Portugal, October 22–24, 2023, Proceedings*, volume 14284. Springer Nature, 2023.

[96] Anders Hessel, Kim G Larsen, Marius Mikucionis, Brian Nielsen, Paul Pettersson, and Arne Skou. Testing real-time systems using uppaal. In *Formal Methods and Testing: An Outcome of the FORTEST Network, Revised Selected Papers*, pages 77–117. Springer, 2008.

[97] Hannes Holm, Martin Karresand, Arne Vidström, and Erik Westring. A survey of industrial control system testbeds. In *Secure IT Systems: 20th Nordic Conference, NordSec 2015, Stockholm, Sweden, October 19–21, 2015, Proceedings*, pages 11–26. Springer, 2015.

[98] David Holmes, Maria Papathanasaki, Leandros Maglaras, Mohamed Amine Ferrag, Surya Nepal, and Helge Janicke. Digital twins and cyber security – solution or challenge? In *2021 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, pages 1–8, 2021.

[99] Gerard J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 1st edition, 2011.

[100] Gerard J. Holzmann and Rajeev Joshi. Model-driven software verification. In *Model Checking Software: 11th International SPIN Workshop, Barcelona, Spain, April 1-3, 2004. Proceedings 11*, pages 76–91. Springer, 2004.

[101] Estelle Hotellier, Franck Sicard, Julien Francq, and Stéphane Mocanu. Standard specification-based intrusion detection for hierarchical industrial control systems. *Information Sciences*, 659:120102, 2024.

[102] Shuang Huang, Chun-Jie Zhou, Shuang-Hua Yang, and Yuan-Qing Qin. Cyber-physical system security for networked industrial processes. *International Journal of Automation and Computing*, 12(6):567–578, 2015.

[103] Marieke Huisman, Corina Păsăreanu, and Naijun Zhan. *Formal Methods: 24th International Symposium, FM 2021, Virtual Event, November 20–26, 2021, Proceedings*. Lecture Notes in Computer Science. Springer International Publishing, 2021.

[104] Eric M. Hutchins, Michael J. Cloppert, Rohan M. Amin, et al. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1(1):80, 2011.

[105] The International Energy Agency (IEA). Iea coal tracking page. Accessed: 12/11/2024.

[106] Security for industrial automation and control systems - part 2-1: Establishing an industrial automation and control systems security program. Standard, European Committee for Electrotechnical Standardization, CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels, November 2010.

[107] Security for industrial automation and control systems - part 3-3: System security requirements and security levels. Standard, European Committee for Electrotechnical Standardization, CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels, August 2013.

[108] Security for industrial automation and control systems - part 4-2: Technical security requirements for iacs components. Standard, European Committee for Electrotechnical Standardization, CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels, April 2019.

[109] D. Iglesias, P. Bunting, S. Esquembri, J. Hollocombe, S. Silburn, L. Vitton-Mea, I. Balboa, A. Huber, G.F. Matthews, V. Riccardo, F. Rimini, and D. Valcarcel. Digital twin applications for the jet divertor. *Fusion Engineering and Design*, 125:71–76, 2017.

[110] Gartner Inc. Gartner information technology glossary: Digital twin, 2024.

[111] InfluxData Inc. InfluxDB OSS. `https://www.influxdata.com/downloads/`, Jan 2024. v2.7.5.

[112] Automation systems and integration — digital twin framework for manufacturing. Standard, International Organization for Standardization, Geneva, CH, March 2021.

[113] Daniel Jackson. Lightweight formal methods. In *FME 2001: Formal Methods for Increasing Software Productivity: International Symposium of Formal Methods Europe Berlin, Germany, March 12–16, 2001 Proceedings*, pages 1–1. Springer, 2001.

[114] Daniel Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on software engineering and methodology (TOSEM)*, 11(2):256–290, 2002.

[115] Daniel Jackson. *Software Abstractions: logic, language, and analysis*. MIT press, 2012.

[116] Eli Jellenc. Explaining politico-strategic cyber security: The feasibility of applying arms race theory. In *Proceedings of the 11th European Conference on Information Warfare and Security*, pages 151–162, 2012.

[117] Eunsuk Kang, Sridhar Adepu, Daniel Jackson, and Aditya P Mathur. Model-based security analysis of a water treatment system. In *Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems*, pages 22–28, 2016.

[118] Hilbert J Kappen. Optimal control theory and the linear bellman equation. 2011.

[119] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, and Jesus Alonso-Zarate. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud computing*, 3(1):11–17, 2015.

[120] Niharika Karnik, Urvi Bora, Karan Bhadri, Prasanna Kadambi, and Pankaj Dhatrak. A comprehensive study on current and future trends towards the characteristics and enablers of industry 4.0. *Journal of Industrial Information Integration*, 27:100294, 2022.

[121] William Kavanagh and Alice Miller. Gameplay analysis of multiplayer games with verified action-costs. *The Computer Games Journal*, 10:89–110, 2021.

[122] William Kavanagh, Alice Miller, Gethin Norman, and Oana Andrei. Balancing turn-based games with chained strategy generation. *IEEE Transactions on Games*, 13(2):113–122, 2019.

[123] Shawkat Sabah Khairullah. Formal verification of a dependable state machine-based hardware architecture for safety-critical cyber-physical systems: Analysis, design, and implementation. *Journal of Electronic Testing*, 40(4):509–523, 2024.

[124] Rafiullah Khan, Peter Maynard, Kieran McLaughlin, David Laverty, and Sakir Sezer. Threat analysis of blackenergy malware for synchrophasor based real-time control and monitoring in smart grid. In *4th International Symposium for ICS & SCADA Cyber Security Research 2016*, pages 53–63. BCS, 2016.

[125] Joseph Migga Kizza. *Software Issues: Risks and Liabilities*, pages 149–176. Springer International Publishing, Cham, 2019.

[126] Michael Kläs and Anna Maria Vollmer. Uncertainty in machine learning applications: A practice-driven classification of uncertainty. In *International Conference on Computer Safety, Reliability, and Security*, pages 431–438. Springer, 2018.

[127] Calvin Ko, Manfred Ruschitzka, and Karl Levitt. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In *Proceedings. 1997 IEEE symposium on security and privacy (Cat. No. 97CB36097)*, pages 175–187. IEEE, 1997.

[128] Sam Kottler, Mehdy Khayamy, Syed Rafay Hasan, and Omar Elkeelany. Formal verification of ladder logic programs using nusmv. In *SoutheastCon 2017*, pages 1–5. IEEE, 2017.

[129] Sam Kottler, Mehdy Khayamy, Syed Rafay Hasan, and Omar Elkeelany. Formal verification of ladder logic programs using nusmv. In *SoutheastCon 2017*, pages 1–5, 2017.

[130] Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihn. Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11):1016–1022, 2018. 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018.

[131] Daniel Kroening and Michael Tautschnig. Cbmc – c bounded model checker. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 389–391, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[132] Thomas Kropf. Introduction to formal hardware verification: Methods and tools for designing correct circuits and systems. 1999.

[133] Maxim Krylov. Free open source historian - setup. `https://kiptr.com/article3_2`. Accessed: 27/11/2024.

[134] Tomas Kulik, Brijesh Dongol, Peter Gorm Larsen, Hugo Daniel Macedo, Steve Schneider, Peter WV Tran-Jørgensen, and James Woodcock. A survey of practical formal methods for security. *Formal Aspects of Computing*, 34(1):1–39, 2022.

[135] Tomas Kulik, Cláudio Gomes, Hugo Daniel Macedo, Stefan Hallerstede, and Peter Gorm Larsen. Towards secure digital twins. In *International Symposium on Leveraging Applications of Formal Methods*, pages 159–176. Springer, 2022.

[136] Tomas Kulik and Peter Gorm Larsen. Towards formal verification of cyber security standards. *Proceedings of the Institute for System Programming of the RAS*, 30:79–94, 10 2018.

[137] Tomas Kulik, Peter W. V. Tran-Jørgensen, and Jalil Boudjadar. Formal security analysis of cloud-connected industrial control systems. In Jean-Louis Lanet and Cristian Toma, editors, *Innovative Security Solutions for Information Technology and Communications*, pages 71–84, Cham, 2019. Springer International Publishing.

[138] Tomas Kulik, Peter W. V. Tran-Jørgensen, Jalil Boudjadar, and Carl Schultz. A framework for threat-driven cyber security verification of iot systems. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 89–97. IEEE, 2018.

[139] Apurva Kumar. A lightweight formal approach for analyzing security of web protocols. In Angelos Stavrou, Herbert Bos, and Georgios Portokalidis, editors, *Research in Attacks, Intrusions and Defenses*, pages 192–211, Cham, 2014. Springer International Publishing.

[140] Marta Kwiatkowska and Gethin Norman. Verifying randomized byzantine agreement. In *FORTE*, volume 2529, pages 194–209, 2002.

[141] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[142] Marta Kwiatkowska, Gethin Norman, and Roberto Segala. Automated verification of a randomized distributed consensus protocol using cadence smv and prism? In *International Conference on Computer Aided Verification*, pages 194–206. Springer, 2001.

[143] Heikki Laaki, Yoan Miche, and Kari Tammi. Prototyping a digital twin for real time remote control over mobile networks: Application of remote surgery. *IEEE Access*, 7:20325–20336, 2019.

[144] Leslie Lamport. Specifying systems: the tla+ language and tools for hardware and software engineers. 2002.

[145] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, 9(3):49–51, 2011.

[146] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. Automatic analysis of a non-repudiation protocol. *Electronic Notes in Theoretical Computer Science*, 112:113–129, 2005.

[147] Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International journal on software tools for technology transfer*, 1:134–152, 1997.

[148] Bruce LeBlanc and Carlos Ferreira. Experimental characterization of h-vawt turbine for development of a digital twin. In *Journal of Physics: Conference Series*, volume 1452, page 012057. IOP Publishing, 2020.

[149] Robert M. Lee, Michael J. Assante, and T Conway. Crashoverride: Analysis of the threat to electric grid operations. *Dragos Inc., March*, page 7, 2017.

[150] Mengnan Liu, Shuiliang Fang, Huiyue Dong, and Cunzhi Xu. Review of digital twin about concepts, technologies, and industrial applications. *Journal of Manufacturing Systems*, 2020.

[151] David F Longbine. Red teaming: past and present. *School of Advanced Military Studies, Army Command and General Staff College*, 2008.

[152] Efrén López-Morales, Carlos Rubio-Medrano, Adam Doupé, Yan Shoshitaishvili, Ruoyu Wang, Tiffany Bao, and Gail-Joon Ahn. Honeyplc: A next-generation honeypot for industrial control systems. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 279–291, 2020.

[153] Y. Lu, A. Miller, C. Johnson, Z. Peng, and T. Zhao. Availability analysis of satellite positioning systems for avaiation using the Prism model checker. In *Proceedings of the 17th International Conference on Computational Science and Engineering (CSE 2014)*, pages 704–713, 2014.

[154] Joshua Lubell, Simon Frechette, Robert Lipman, Frederick Proctor, John Horst, Mark Carlisle, and Paul Huang. Model based enterprise summit report. 12 2012.

[155] Feng Luo, Yifan Jiang, Jiajia Wang, Zhihao Li, and Xiaoxian Zhang. A framework for cybersecurity requirements management in the automotive domain. *Sensors*, 23(10), 2023.

[156] Nuwan Sri Madusanka, Yijie Fan, Shaolong Yang, and Xianbo Xiang. Digital twin in the maritime domain: A review and emerging trends. *Journal of Marine Science and Engineering*, 11(5):1021, 2023.

[157] Paolo Maggi and Riccardo Sisto. Using spin to verify security properties of cryptographic protocols. In *International SPIN Workshop on Model Checking of Software*, pages 187–204. Springer, 2002.

[158] Otto Mayr. The origins of feedback control. *Scientific American*, 223(4):110–119, 1970.

[159] Hu Mengyan, Zhang Xueyan, Peng Cuiting, Zhang Yixuan, and Yang Jun. Current status of digital twin architecture and application in nuclear energy field. *Annals of Nuclear Energy*, 202:110491, 2024.

[160] Francesco Mercaldo, Fabio Martinelli, and Antonella Santone. Real-time scada attack detection by means of formal methods. In *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 231–236, 2019.

[161] Stevan A. Milinković and Ljubomir R. Lazić. Industrial plc security issues. In *2012 20th Telecommunications Forum (TELFOR)*, pages 1536–1539, 2012.

[162] A. Miller and M. Calder. An application of abstraction and induction techniques to degenerating systems of processes. In *Proceedings of the International Workshop on Model-Checking for Dependable Software-Intensive Systems (MCDSIS'03), supplement to the*

Proceedings of DSN-2003, pages W–75–W–79, San Francisco, June 2003. IEEE Computer Society Press.

[163] A. Miller, A. F. Donaldson, and M. Calder. Symmetry in temporal logic model checking. *ACM Computing Surveys*, 38(3), 9 2006.

[164] Alice Miller, Bernd Porr, Ivaylo Valkov, Douglas Fraser, and Daumantas Pagojus. Model checking with memoisation for close to real-time overtaking planning. In *Science of Computer Programming*, 2025. Accepted.

[165] Robert Mitchell and Ing-Ray Chen. Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems. *IEEE Transactions on Dependable and Secure Computing*, 12(1):16 – 30, 2015.

[166] Abhijit Mohanta and Anoop Saldanha. *Static Analysis*, pages 377–402. Apress, Berkeley, CA, 2020.

[167] Andrés Murillo, Riccardo Taormina, Nils Ole Tippenhauer, Davide Salaorni, Robert van Dijk, Luc Jonker, Simcha Vos, Maarten Weyns, and Stefano Galelli. High-fidelity cyber and physical simulation of water distribution systems. i: Models and data. *Journal of Water Resources Planning and Management*, 149(5):04023009, 2023.

[168] Roberto Nardone, Ugo Gentile, Adriano Peron, Massimo Benerecetti, Valeria Vittorini, Stefano Marrone, Renato De Guglielmo, Nicola Mazzocca, and Luigi Velardi. Dynamic state machines for formalizing railway control system specifications. In *Formal Techniques for Safety-Critical Systems: Third International Workshop, FTSCS 2014, Luxembourg, November 6-7, 2014. Revised Selected Papers 3*, pages 93–109. Springer, 2015.

[169] Roberto Nardone, Ricardo J. Rodríguez, and Stefano Marrone. Formal security assessment of modbus protocol. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 142–147, 2016.

[170] Amir Nedaei, Mohammadreza Aghaei, Aref Eskandari, Frédéric Maurer, and Umit Cali. Digital twins for hydropower applications. In *Digital Twin Technology for the Energy Sector*, pages 213–234. Elsevier, 2025.

[171] Gethin Norman and Vitaly Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6):561–589, 2006.

[172] Livinus Obiora Nweke. A survey of specification-based intrusion detection techniques for cyber-physical systems. 2021.

[173] International Society of Automation. ISA/IEC 62443 series of standards. `https://www.isa.org/standards-and-publications/isa-standards/isa-iec-62443-series-of-standards`. Accessed: 16/01/2025.

[174] National Institute of Standards and Technology. Framework for improving critical infrastructure cybersecurity. Technical report, National Institute of Standards and Technology, 2 2014. Version 1.

[175] National Institute of Standards and Technology. The nist cybersecurity framework (CSF) 2.0. Technical report, National Institute of Standards and Technology, 2 2024.

[176] OpenJS Foundation Contributors. Node-RED. `https://github.com/node-red/node-red/releases/tag/3.1.3`, Dec 2023. v3.1.3.

[177] Gerard O'Regan. *Verification of Safety-Critical Systems*, pages 235–250. Springer International Publishing, Cham, 2019.

[178] Tolga Ovatman, Atakan Aral, Davut Polat, and Ali Osman Ünver. An overview of model checking practices on verification of plc software. *Software & Systems Modeling*, 15(4):937–960, 2016.

[179] Suhas Pai, Yash Sharma, Sunil Kumar, Radhika M Pai, and Sanjay Singh. Formal verification of oauth 2.0 using alloy framework. In *2011 International Conference on Communication Systems and Network Technologies*, pages 655–659. IEEE, 2011.

[180] Z. Peng, Y. Lu, A. Miller, C. Johnson, and T. Zhao. A probabilistic model checking approach to analysing reliability, availability, and maintainability of a single satellite system. In *Proceedings of the 7th UKSim/AMSS European symposium on Computer Modelling and Simulation (EMS2013)*, pages 573–578, Manchester, England, November 2013. IEEE Computer Society Press.

[181] Alberto Pettorossi and Maurizio Proietti. *Program Derivation = Rules + Strategies*, pages 273–309. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[182] R Piascik, J Vickers, D Lowry, S Scotti, J Stewart, and A Calomino. Technology area 12: Materials, structures, mechanical systems, and manufacturing road map. *NASA Office of Chief Technologist*, pages 15–88, 2010.

[183] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13(1):45–60, 1981.

[184] Christos Pylianidis, Sjoukje Osinga, and Ioannis N. Athanasiadis. Introducing digital twins to agriculture. *Computers and Electronics in Agriculture*, 184:105942, 2021.

[185] Qinglin Qi and Fei Tao. Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison. *Ieee Access*, 6:3585–3593, 2018.

[186] Shafiqur Rehman, Luai M. Al-Hadhrami, and Md. Mahbub Alam. Pumped hydro energy storage system: A technological review. *Renewable and Sustainable Energy Reviews*, 44:586–598, 2015.

[187] Mark Reynolds and Azer Bestavros. Formal verification of cross-domain access control policies using model checking. Technical report, Computer Science Department, Boston University, 2011.

[188] Marco Rocchetto and Nils Ole Tippenhauer. On attacker models and profiles for cyber-physical systems. In Ioannis Askoxylakis, Sotiris Ioannidis, Sokratis Katsikas, and Catherine Meadows, editors, *Computer Security – ESORICS 2016*, pages 427–449, Cham, 2016. Springer International Publishing.

[189] Marco Rocchetto and Nils Ole Tippenhauer. Towards formal security analysis of industrial control systems. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 114–126, 2017.

[190] Roland Rosen, Georg von Wichert, George Lo, and Kurt D. Bettenhausen. About the importance of autonomy and digital twins for the future of manufacturing. *IFAC-PapersOnLine*, 48(3):567–572, 2015. 15th IFAC Symposium onInformation Control Problems inManufacturing.

[191] Juan E Rubio, Rodrigo Roman, and Javier Lopez. Analysis of cybersecurity threats in industry 4.0: the case of intrusion detection. In *Critical Information Infrastructures Security: 12th International Conference, CRITIS 2017, Lucca, Italy, October 8-13, 2017, Revised Selected Papers 12*, pages 119–130. Springer, 2018.

[192] Kousei Sakata, Shintaro Fujita, Kenji Sawada, Hiroshi Iwasawa, Hiromichi Endoh, and Noritaka Matsumoto. Model verification of fallback control system under cyberattacks via uppaal. *Advanced Robotics*, 37(3):156–168, 2023.

[193] K. Schwab and World Economic Forum. *The Fourth Industrial Revolution*. World Economic Forum, 2016.

[194] Klaus Schwab. The fourth industrial revolution: what it means, how to respond. In *Handbook of research on strategic leadership in the Fourth Industrial Revolution*, pages 29–34. Edward Elgar Publishing, 2024.

[195] Anne Seegrün, Thomas Kruschke, Janine Mügge, Louis Hardinghaus, Tobias Knauf, Theresa Riedelsheimer, and Kai Lindow. Sustainable product lifecycle management with

digital twins: A systematic literature review. *Procedia CIRP*, 119:776–781, 2023. The 33rd CIRP Design Conference.

[196] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: a new approach for detecting network intrusions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, page 265–274, New York, NY, USA, 2002. Association for Computing Machinery.

[197] Concetta Semeraro, Mario Lezoche, Hervé Panetto, and Michele Dassisti. Digital twin paradigm: A systematic literature review. *Computers in Industry*, 130:103469, 2021.

[198] Mike Shafto, Michael Conroy, R Doyle, E Glaessgen, C Kemp, J LeMoigne, and L Wang. Technology area 11: Modeling, simulation, information technology and processing roadmap, 05 2010.

[199] Roshan Shrestha, Hoda Mehrpouyan, and Dianxiang Xu. Model checking of security properties in industrial control systems (ics). In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, CODASPY '18, page 164–166, New York, NY, USA, 2018. Association for Computing Machinery.

[200] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. Industrial internet of things: Challenges, opportunities, and directions. *IEEE transactions on industrial informatics*, 14(11):4724–4734, 2018.

[201] Krishnamoorthi Sivalingam, Marco Sepulveda, Mark Spring, and Peter Davies. A review and methodology development for remaining useful life prediction of offshore fixed and floating wind turbine power converter with digital twin technology perspective. In *2018 2nd International Conference on Green Energy and Applications (ICGEA)*, pages 197–204, 2018.

[202] Joe Slowik. Anatomy of an attack: Detecting and defeating crashoverride. *VB2018, October*, 2018.

[203] Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, and Prateek Saxena. Bitblaze: A new approach to computer security via binary analysis. In *Information Systems Security: 4th International Conference, ICISS 2008, Hyderabad, India, December 16-20, 2008. Proceedings 4*, pages 1–25. Springer, 2008.

[204] Tao Song, Calvin Ko, Chinyang Henry Tseng, Poornima Balasubramanyam, Anant Chaudhary, and Karl N Levitt. Formal reasoning about a specification-based intrusion

detection for dynamic auto-configuration protocols in ad hoc networks. In *Formal Aspects in Security and Trust: Thrid International Workshop, FAST 2005, Newcastle upon Tyne, UK, July 18-19, 2005, Revised Selected Papers*, pages 16–33. Springer, 2006.

[205] V Stafford. Zero trust architecture. *NIST Special Publication 800-207*, 800:207, 2020.

[206] Mateusz Stafiniak and Wojciech Wodo. State-sponsored cybersecurity attacks. In *2022 63rd International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS)*, pages 1–6. IEEE, 2022.

[207] Susan Stepney, David Cooper, and Jim Woodcock. An electronic purse: Specification, refinement and proof. 2000.

[208] Keith Stouffer, Michael Pease, CheeYee Tang, Timothy Zimmerman, Victoria Pillitteri, Suzanne Lightman, Adam Hahn, Stephanie Saravia, Aslam Sherule, and Michael Thompson. Guide to operational technology (OT) security. Technical report, National Institute for Standards and Technology, 2023.

[209] Tianze Sun, Xiwang He, and Zhonghai Li. Digital twin in healthcare: Recent updates and challenges. *Digital Health*, 9:20552076221149651, 2023.

[210] Fei Tao, Jiangfeng Cheng, Qinglin Qi, Meng Zhang, He Zhang, and Fangyuan Sui. Digital twin-driven product design, manufacturing and service with big data. *The International Journal of Advanced Manufacturing Technology*, 94:3563–3576, 2018.

[211] Fei Tao, Qinglin Qi, Lihui Wang, and AYC Nee. Digital twins and cyber–physical systems toward smart manufacturing and industry 4.0: Correlation and comparison. *Engineering*, 5(4):653–661, 2019.

[212] Ankit Thakkar and Ritika Lohiya. A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions. *Artificial Intelligence Review*, 55(1):453–563, 2022.

[213] Tej Narayan Thakur and Noriaki Yoshiura. Modeling and verification of contactless mobile banking system in e-banking using spin. In *Computational Science and Its Applications–ICCSA 2021: 21st International Conference, Cagliari, Italy, September 13–16, 2021, Proceedings, Part VI 21*, pages 581–597. Springer, 2021.

[214] Office of the Press Secretary The White House. Executive order – improving critical infrastructure cybersecurity, 2 2013.

[215] Jean-Charles Tournier, Borja Fernández Adiego, and Ignacio D. Lopez-Miguel. PLCverif: Status of a formal verification tool for programmable logic controller. *Pro-*

ceedings of the 18th International Conference on Accelerator and Large Experimental Physics Control Systems, ICALEPCS2021:China, 2022.

[216] Peter W. V. Tran-Jørgensen, Tomas Kulik, Jalil Boudjadar, and Peter Gorm Larsen. Security analysis of cloud-connected industrial control systems using combinatorial testing. In *Proceedings of the 17th ACM-IEEE International Conference on Formal Methods and Models for System Design*, pages 1–11, 2019.

[217] Phu Truong, Dennis Nieh, and Melody Moh. Specification-based intrusion detection for h. 323-based voice over ip. In *Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology, 2005.*, pages 387–392. IEEE, 2005.

[218] Giannis Tsaraias and Ivan Speziale. Industroyer vs. industroyer2: Evolution of the iec 104 component. *Industroyer2: Evolution of the IEC 104 Component*, 2022.

[219] Chin-Yang Tseng, Poornima Balasubramanyam, Calvin Ko, Rattapon Limprasittiporn, Jeff Rowe, and Karl Levitt. A specification-based intrusion detection system for aodv. In *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 125–134, 2003.

[220] Nikolaos Tzanis, Nikolaos Andriopoulos, Aris Magklaras, Eleftherios Mylonas, Michael Birbas, and Alexios Birbas. A hybrid cyber physical digital twin approach for smart grid fault prediction. In *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, volume 1, pages 393–397. IEEE, 2020.

[221] Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei, and István Majzik. Theta: A framework for abstraction refinement-based model checking. In *2017 Formal Methods in Computer Aided Design (FMCAD)*, pages 176–179, 2017.

[222] Muhammad Azmi Umer, Khurum Nazir Junejo, Muhammad Taha Jilani, and Aditya P. Mathur. Machine learning for intrusion detection in industrial control systems: Applications, challenges, and recommendations. *International Journal of Critical Infrastructure Protection*, 38:100516, 2022.

[223] Prem Uppuluri and R Sekar. Experiences with specification-based intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 172–189. Springer, 2001.

[224] Hendrik van der Valk, Hendrik Haße, Frederik Möller, Michael Arbter, Jan-Luca Henning, and Boris Otto. A taxonomy of digital twins. In *Proc. 26th Americas Conference on Information Systems*, pages 1–10, 2020.

[225] David von Oheimb and Sebastian Mödersheim. Aslan++ — a formal security specification language for distributed systems. In Bernhard K. Aichernig, Frank S. de Boer, and Marcello M. Bonsangue, editors, *Formal Methods for Components and Objects*, pages 1–22, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[226] DJ Wagg, P Gardner, RJ Barthorpe, and K Worden. On key technologies for realising digital twins for structural dynamics applications. In *Model Validation and Uncertainty Quantification, Volume 3*, pages 267–272. Springer, 2020.

[227] L. Wang and F. Cai. Reliability analysis for flight control systems using probabilistic model checking. *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, 2017-Novem:161–164, 2017.

[228] Rui Wang, Yong Guan, Houbing Song, Xinxin Li, Xiaojuan Li, Zhiping Shi, and Xiaoyu Song. A formal model-based design method for robotic systems. *IEEE Systems Journal*, 13(1):1096–1107, 2019.

[229] Ting Wang, Qi Su, and Tieming Chen. Formal analysis of security properties of cyber-physical system based on timed automata. In *2017 IEEE Second International Conference on Data Science in Cyberspace (DSC)*, pages 534–540. IEEE, 2017.

[230] Ting Wang, Qi Su, and Tieming Chen. Formal analysis of security properties of cyber-physical system based on timed automata. In *2017 IEEE Second International Conference on Data Science in Cyberspace (DSC)*, pages 534–540, 2017.

[231] Weiguang Wang, Qingkai Zeng, and Aditya P Mathur. A security assurance framework combining formal verification and security functional testing. In *2012 12th International Conference on Quality Software*, pages 136–139. IEEE, 2012.

[232] Xiaoliang Wang and Adrian Rutle. Model checking healthcare workflows using alloy. *Procedia Computer Science*, 37:481–488, 2014.

[233] Yuntao Wang, Zhou Su, Shaolong Guo, Minghui Dai, Tom H Luan, and Yiliang Liu. A survey on digital twins: Architecture, enabling technologies, security and privacy, and future prospects. *IEEE Internet of Things Journal*, 10(17):14965–14987, 2023.

[234] Zibo Wang, Yaofang Zhang, Yilu Chen, Hongri Liu, Bailing Wang, and Chonghua Wang. A survey on programmable logic controller vulnerabilities, attacks, detections, and forensics. *Processes*, 11(3), 2023.

[235] Haroon Wardak, Sami Zhioua, and Ahmad Almulhem. Plc access control: a security analysis. In *2016 World Congress on Industrial Control Systems Security (WCICSS)*, pages 1–6, 2016.

[236] Qianxin Wei, Jian Jiao, and Tingdi Zhao. Flight control system failure modeling and verification based on spin. *Engineering failure analysis*, 82:501–513, 2017.

[237] Markus Weißmann, Stefan Bedenk, Christian Buckl, and Alois Knoll. Model checking industrial robot systems. In *Model Checking Software: 18th International SPIN Workshop, Snowbird, UT, USA, July 14-15, 2011. Proceedings 18*, pages 161–176. Springer, 2011.

[238] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995.

[239] Theodore J Williams. The purdue enterprise reference architecture. *Computers in industry*, 24(2-3):141–158, 1994.

[240] Bradley J Wood and Ruth A Duggan. Red teaming of advanced information assurance concepts. In *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, volume 2, pages 112–118. IEEE, 2000.

[241] Zeng Xiangjun, Yang Ming, Yang Xianglong, Bo Yifan, Feng Chen, and Zhou Yu. Anomaly detection of wind turbine gearbox based on digital twin drive. In *2020 IEEE 3rd Student Conference on Electrical Machines and Systems (SCEMS)*, pages 184–188, 2020.

[242] D. Yadron and D. Tynan. Tesla driver dies in first fatal crash while using autopilot mode. *The Guardian*, 2016. Retrieved from `https://www.theguardian.com/technology/2016/jun/30/tesla-autopilot-death-self-driving-car-elon-musk`.

[243] Muhammad Mudassar Yamin, Basel Katt, and Vasileios Gkioulos. Cyber ranges and security testbeds: Scenarios, functions, tools and architecture. *Computers Security*, 88:101636, 2020.

[244] Huan Yang, Liang Cheng, and Mooi Choo Chuah. Deep-learning-based network intrusion detection for scada systems. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pages 1–7, 2019.

[245] Wei Yu, Panos Patros, Brent Young, Elsa Klinac, and Timothy Gordon Walmsley. Energy digital twin technology for industrial energy management: Classification, challenges and future. *Renewable and Sustainable Energy Reviews*, 161:112407, 2022.

[246] Luca Zanolin, Carlo Ghezzi, and Luciano Baresi. An approach to model and validate publish/subscribe architectures. In *Proc. of the SAVCBS*, volume 3, pages 35–41, 2003.

[247] Wei Zhang, Wenke Ma, Huiling Shi, and Fu-qiang Zhu. Model checking and verification of the internet payment system with spin. *J. Softw.*, 7(9):1941–1949, 2012.

[248] Zixuan Zhao, Hui Li, and Qiang Fu. A digital twin platform for the industrial control system of chemical production. In *Chinese Intelligent Systems Conference*, pages 151–158. Springer, 2024.

[249] Giulio Zizzo, Chris Hankin, Sergio Maffeis, and Kevin Jones. Adversarial machine learning beyond the image domain. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[250] Saman Zonouz, Julian Rrushi, and Stephen McLaughlin. Detecting industrial control malware using automated plc code analytics. *IEEE Security & Privacy*, 12(6):40–47, 2014.

Appendix A

Supporting Code and Models

An archive of code and models used in the creation of the thesis can be accessed at: `doi.org/10.5281/zenodo.15482551`

This repository includes:

- The trunk and branch Promela model templates used in Chapter 5
- DT module used to connect to InfluxDB database and interface data to worker threads executing the SPIN model checker.
- Test scripts used to execute evaluation of SPIN models.
- A series of Alloy models documented in Chapter 6.
- Documentation for running the models.