CYPRESS

CYPRESS: Report D3.2 - Enhancing Cyber-Physical Security in Real-Time Operation

Task 3.2 - Project Report

Mathy Laurent, Benoît Knott, Tohid Behdadnia, Dave Singelée, Can Özkan, Xinhai Zhou, Shahabeddin Kamyab

Date: 2025-01-14

Contents

Executive Summary

The CYPRESS project focuses on addressing the Cyber-Physical Risk associated with the bulk Electric Energy Supply (EES) system, which is becoming increasingly complex due to the growing integration of information and communication technologies. This transformation brings significant benefits but also introduces novel vulnerabilities, particularly in terms of cyber-threats. The overarching aim of CYPRESS is to develop the knowledge, methods, and tools necessary to ensure the security of electric energy supply systems, accounting for these cyber-physical risks and integrating them into a coherent probabilistic risk management framework. To achieve this, the project is structured around three main research themes: i) the development of novel models and benchmarks for the simulation and testing of cyber-physical systems, ii) the assessment of these systems' vulnerabilities, and iii) the enhancement of their resilience to threats.

Work Package 3 (WP3), titled "Mitigation of cyber-physical security risks," is a key part of this effort. WP3's main objective is to develop strategies and decision-making tools to help reduce the risks associated with cyber-physical vulnerabilities in power systems. Within WP3, Task 3.2 (T3.2) specifically targets the development of real-time operation strategies that incorporate cyber-threats into power system operations, providing tools and techniques for the immediate detection, assessment, and mitigation of such threats.

The T3.2 sub-task investigates decision-making aids for real-time operations, extending traditional power system operational frameworks to include cyber-physical considerations. This includes strategies for system observability, monitoring cyber-physical interactions, and ensuring resilient recovery from attacks. Methods developed in T3.2 are built on previous work in preventive and corrective risk management and are designed to be implemented in the transmission system operator (TSO) and distribution system operator (DSO) environments. These methods will allow for the continuous monitoring and response to cyber-physical risks as they arise, optimizing both preventive measures and corrective actions in line with Objectives 1-3, while Objective 4 focuses on ensuring the resilience of cyber-physical infrastructures even when partially compromised, through novel design, algorithms, and protocols.

The document is structured as follows:

- Chapter 1: Introduction
- Chapter 2: Diagnostic Agents and Cyber-Space Observability (Objective 1)
- Chapter 3: IT Anomaly Detection (Objective 2)
- Chapter 4: Cyber-Operator Characterization (Objective 3)
- Chapter 5: Design of Cyber-Attack Resilient Cyber-Physical Systems (Objective 4)
- Chapter 6: Conclusion

This deliverable underlines significant advancements in enhancing cybersecurity resilience, encompassing real-time cyber-space observability, robust IT anomaly detection, comprehensive cyber-operator characterization,

and innovative approaches for designing cyber-attack resilient systems. It introduces frameworks and techniques that proactively address vulnerabilities, strengthen operational integrity, and provide practical solutions to safeguard cyber-physical infrastructures.

The insights gained through T3.2 are expected to contribute to enhancing the resilience and security of future power systems, providing valuable tools for operators to mitigate cyber-physical risks in real-time and under uncertain conditions.

Author Contributions

Author	Affiliation
Laurent Mathy	Université de Liège
Benoît Knott	Université de Liège
Tohid Behdadnia	KU Leuven
Dave Singelée	KU Leuven
Can Özkan	KU Leuven
Xinhai Zhou	KU Leuven
Shahabeddin Kamyab	Université Libre de Bruxelles

Table 1: List of authors

List of acronyms

- ACL: Access Control List
- AI: Artificial Intelligence
- AIBOM: AI Bill of Materials
- APT: Advanced Persistent Threat
- AWIPS: Advanced Weather Interactive Processing System
- BC-SCM: Blockchain-based Supply Chain Management
- BOM: Bill of Materials
- CA: Certification Authority
- CHIRP: CISA Hunt and Incident Response Program
- CISA: Cybersecurity and Infrastructure Security Agency
- CLOVE: Congestion-Aware Load Balancing at the Virtual Edge
- CNN: Convolutional Neural Network
- CPPS: Cyber-Physical Power System
- CPS: Cyber-Physical System
- CSaS: Cyber-Space as Services
- CSOD: Cyber-Space Observability through Dependency Graph
- CVE: Common Vulnerabilities and Exposures
- CVSS: Common Vulnerability Scoring System
- DCN: Data Center Network
- DNP: Distributed Network Protocol
- DSO: Distribution System Operator
- DX: Data Exchange
- ECDSA: Elliptic Curve Digital Signature Algorithm
- ECMP: Equal-Cost Multi-Path
- EES: Electric Energy Supply

- ESP: Encapsulating Security Payload
- EU: European Union
- FCT: Flow Completion Time
- FFT: Fast Fourier Transform
- FIX: Financial Information Exchange
- FS: File System
- GAN: Generative Adversarial Network
- GAT: Gitlab Advisory Tool
- gRPC: gRPC Remote Procedure Calls
- ICT: Information and Communication Technologies
- ID: Identifier
- IEC: International Electrotechnical Commission
- IEEE: Institute of Electrical and Electronics Engineers
- IIDG: Inter-Instance Dependency Graph
- INT: In-band Network Telemetry
- IP: Internet Protocol
- IT: Information Technology
- JNDI: Java Naming and Directory Interface
- JS: JavaScript
- LAN: Local Area Network
- LDAP: Lightweight Directory Access Protocol
- LPCMCI: Linear Prediction with Momentary Conditional Independence
- LSTM: Long Short-Term Memory
- MAC: Media Access Control
- MAC: Message Authentication Code
- ML: Machine Learning
- MODIS: Moderate Resolution Imaging Spectroradiometer
- NEXRAD: Next Generation (Weather) Radar
- NI: National Instruments
- NTIA: National Telecommunications and Information Administration
- NTS: Network Traffic Signal
- NVD: National Vulnerability Database
- NVMe: Non-Volatile Memory Express
- OPC: Open Platform Communications
- OSI: Open Systems Interconnection
- OS: Operating System
- OSV: Open Source Vulnerabilities
- OWASP: Open Worldwide Application Security Project
- PMU: Phasor Measurement Unit

- POSIX: Portable Operating System Interface
- RS: Recommended Standard
- RTP: Real-time Transport Protocol
- RTSP: Real-time Streaming Protocol
- RTT: Round-Trip Time
- RTU: Remote Terminal Unit
- SBOM: Software Bill of Materials
- SCADA: Supervisory Control And Data Acquisition
- SCA: Software Composition Analysis
- SDLC: Software Development Life Cycle
- SDN: Software Defined Networking
- SHA: Secure Hash Algorithm
- SLA: Service Level Agreement
- SPDX: Software Package Data Exchange
- SRv6: Segment Routing over IPv6
- SSD: Solid-State Disk
- SSH: Secure Shell
- SSL: Secure Sockets Layer
- STRIDE: Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, Elevation of privilege
- SV: Sampled Value
- TLS: Transport Layer Security
- tsFCI: time series Fast Causal Inference
- TSO: Transmission System Operator
- USB: Universal Serial Bus
- VEX: Vulnerability-Exploitability eXchange
- VFS: Virtual File System
- VLAN: Virtual Local Area Network
- VPN: Virtual Private Network
- WAL: Write-Ahead Logging
- WAN: Wide-Area Network
- WP: Work Package

List of Figures

List of Tables

1

Introduction

The modernization of electric power systems has led to their transformation into highly interconnected and complex cyber-physical systems. These systems integrate traditional physical infrastructures with advanced information and communication technologies (ICT), enabling smarter, more flexible, and efficient energy management. However, this evolution comes with significant challenges, as the same interconnectedness and technological sophistication expose power systems to a broad array of vulnerabilities, particularly in the cyber domain. Cyber-threats, ranging from data breaches to coordinated attacks on critical infrastructure, pose a severe risk to the secure and reliable operation of these systems. Ensuring the resilience and security of the electric power grid has therefore become a top priority, especially for the bulk Electric Energy Supply (EES) system, which forms the backbone of modern energy distribution networks.

The CYPRESS project, focused on the Cyber-Physical Risk of the bulk Electric Energy Supply system, addresses these critical challenges by developing innovative methods, tools, and frameworks to enhance the security and resilience of the transmission grid. Its goal is to integrate the unique characteristics of cyber-physical threats into a comprehensive probabilistic risk management approach, combining state-of-the-art modeling, assessment, and control techniques. By doing so, CYPRESS aims to provide actionable solutions for detecting, preventing, and mitigating cyber-attacks on power systems, while also equipping operators with decision-making tools to maintain continuity under adverse conditions.

This deliverable, D3.2, is part of Work Package 3 (WP3), titled "Mitigation of Cyber-Physical Security Risks", which focuses on developing methodologies to reduce vulnerabilities in the power system through advanced strategies for real-time operation. Specifically, Task 3.2 (T3.2) explores how diagnostic tools, anomaly detection mechanisms, and cyber-resilience frameworks can be integrated into the real-time operational landscape of power systems. The outcomes presented in this deliverable contribute directly to enhancing situational awareness, optimizing incident response strategies, and ensuring resilience against both known and emerging cyber-physical threats.

1.1 Scope and Structure of the Document

The document is structured to present a holistic approach to mitigating cyber-physical risks in power systems through a combination of innovative techniques and decision-making aids. The Executive Summary provides a concise overview of the key findings and their implications for the real-time security of power systems.

The deliverable is organized around four primary objectives, aligned with the broader goals of the CYPRESS project:

1. Diagnostic Agents and Cyber-Space Observability (Chapter 2): This section discusses the development of tools for extracting state variables from cyber-infrastructures, enabling enhanced monitoring of real-time performance and security. By introducing measurable states into cyber-systems, this work aims to bring transparency to otherwise opaque environments.
2. IT Anomaly Detection (Chapter 3): Here, the focus is on identifying deviations from normal operational patterns in ICT systems. Detecting anomalies is critical for raising timely alarms and mitigating potential intrusions before they escalate.
3. Cyber-Operator Characterization (Chapter 4): This chapter explores the decision-making landscape of cyber-operators, identifying the preventive and corrective actions they can take in response to cyber-threats. It also examines the factors influencing these decisions, contributing to a better understanding of human-cyber interaction in high-stakes environments.
4. Design of Cyber-Attack Resilient Cyber-Physical Systems (Chapter 5): This section proposes strategies for ensuring that power systems can maintain their functionality even when the underlying ICT infrastructure is partially compromised. By designing resilient architectures, the impact of cyber-attacks can be minimized.

Finally, the Conclusion (Chapter 6) summarizes the key contributions of this deliverable and outlines potential avenues for future research within the framework of the CYPRESS project.

By addressing these objectives, this deliverable contributes to the overarching vision of the CYPRESS project: to ensure the secure and resilient operation of cyber-physical power systems in an increasingly digital and interconnected world.

2

Diagnostic Agents and Cyber-Space Observability (Objective 1)

2.1 Introduction

The rapid evolution of the electrical grid into a cyber-physical system (CPS) intertwines traditional power infrastructures with complex digital layers, creating a hybrid environment where resilience and security rely on the synergy between these domains. Cyber-Space Observability, as addressed in CYPRESS's Objective 1, is foundational to enhancing diagnostic capabilities across such systems, aiming to achieve real-time visibility and operational assurance in this cyber-physical landscape. Specifically, Objective 1 seeks to develop diagnostic agents capable of extracting measurable state variables from the digital components of the electrical network, introducing a layer of observability traditionally absent from these cyber structures.

Observability here refers to the capability to infer the internal states of a system through external outputs, often using key indicators like metrics, logs, and traces. In cyber-physical systems, this observability becomes critical to maintaining a resilient infrastructure, as it enables operators to assess and respond to system vulnerabilities effectively. Objective 1 leverages observability principles by modeling diagnostic agents to continuously monitor and quantify the performance of cyber-infrastructure in real time. This allows proactive identification of potential risks, facilitates optimization of system performance, and supports predictive maintenance frameworks.

Under CYPRESS's Task 3.2, Objective 1 aligns closely with diagnostic strategies developed in Work Package 1 (WP1), which defines real-time testing criteria and provides reliability indicators (T1.1), key network components and threats (T1.2), and benchmarks and assessments (T1.3) for cyber-physical infrastructure. Building on WP1's insights, the Diagnostic Agent (DxAgent) and the Cyber-Space Observability through Dependency Graph (CSOD) framework provide a structured approach to observability, modeling cyber-infrastructure as interdependent service layers that allow for precise, quantifiable analysis. This layered model not only provides critical insights into potential failure cascades within the network but also identifies core vulnerabilities where added

redundancies can mitigate risks effectively. Additionally, WP3 is expected to improve WP1's reliability indicators as part of this collaborative exchange.

The development of diagnostic agents draws on foundational concepts established in the DXAgent, which served as an early model for cyber-space observability. DXAgent's focus on real-time diagnostic capabilities through continuous monitoring of cyber-infrastructure has influenced the design of advanced frameworks like SMOOTHIE, which applies similar principles to enhance congestion management and observability in data center environments. This lineage highlights the project's progression in observability techniques, from early diagnostic agents to more complex load-balancing and telemetry-based frameworks. Together, these efforts form a comprehensive observability framework designed to support the safe and resilient operation of the electrical grid's increasingly digital infrastructure.

2.2 DxAgent (Sami Ben Mariem, Vincent Rossetto)

The DxAgent, as proposed in CYPRESS's Cyber-Space Observability framework, serves as a core diagnostic component for monitoring the state of the cyber-infrastructure in real-time. It is a distributed entity designed to provide operators with a rigorous, summarized view of the system's health and status. Leveraging principles from the Cyber-Space as a Service (CSaS) model, DxAgent operates by aggregating cyber-components into higher-level services. These services, evaluated through real-time metrics, offer an accessible, high-level summary of the infrastructure's operational state, which is critical for informed decision-making and proactive infrastructure management.

2.2.1 Principles and Goals of DxAgent

The DxAgent framework is grounded in several core principles of observability, a key concept in ensuring the resilience of cyber-physical systems. Observability refers to the ability to infer the internal state of a system based on its outputs and behaviors. As shown in Figure 2.1, the DxAgent enables this capability by building an interdependency graph of CSaS-defined services, allowing for ongoing assessment through key indicators like latency, accuracy, availability, and resilience. This graph forms the basis for real-time monitoring and presents operators with a summarized, yet comprehensive view of the infrastructure.

The DxAgent framework is designed to:

- Assist operators in both short-term and long-term resource allocation planning by providing insight into the current and predicted state of the cyber-infrastructure.
- Enable dynamic system reconfiguration to address potentially insecure or unstable conditions by identifying critical vulnerabilities in real-time.
- Serve as a foundation for automated decision-making by detecting deviations from the expected system behavior, enhancing response to threats before they escalate into major disruptions.

2.2.2 Challenges in Real-Time Observability

Implementing real-time observability poses notable challenges, particularly in complex and encrypted cyber-infrastructure environments. The DxAgent must derive an accurate interdependency graph from real-time data, an effort complicated by encrypted traffic and the need for efficient measurement techniques. Balancing passive and active measurement strategies further complicates real-time observability, as each approach introduces trade-offs in terms of accuracy, timeliness, and computational cost.

2.2.3 Foundation for Further Development

The DxAgent's ability to provide high-level, real-time insights into cyber-infrastructure status lays the groundwork for the advanced observability approaches discussed in subsequent sections. By offering a clear and actionable view of the current state and dependencies of critical services, the DxAgent not only informs immediate operational decisions but also contributes to long-term resilience strategies through continuous monitoring and risk assessment capabilities.

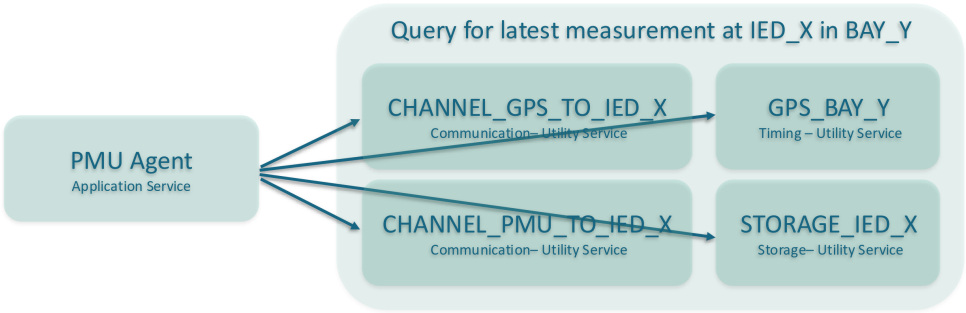


Figure 2.1: Conceptual structure of the DxAgent framework based on CSaS principles.

2.3 CSOD (CSaS) observability framework (Sami Ben Mariem, Vincent Rossetto, François Straet)

The Cyber-Space Observability through Dependency Graph (CSOD) framework provides a systematic approach for modeling and assessing the interdependencies of services in complex cyber-infrastructures. CSOD (formerly referred to as CSaS) utilizes a dependency graph to represent hierarchical relationships between services, allowing for an analysis of how service-level degradations propagate within the infrastructure. Built upon dependency graph metrics as outlined in deliverable D1.2, this model organizes cyber services into layered abstractions, from high-level services down to more granular instances, with dependencies reflecting how a disruption in one component can impact others.

2.3.1 Modeling Service Dependencies

At the core of the CSOD framework is the dependency graph, which quantifies the relationships between services based on the observed adequacy of their outputs relative to baseline Service Level Agreements (SLAs). A service instance S_i is said to depend on another instance S_j if the degradation in S_j's performance metrics, such as latency or availability, could potentially lead to a reduced quality of service in S_i as well. This dependency hierarchy allows the infrastructure to be represented as a directed acyclic graph $G(V, E)$, where each node V is an instance of a service, and each edge E denotes a dependency between two instances.

To achieve a quantitative understanding of these dependencies, CSOD employs a set of SLA metrics: latency, throughput, accuracy, and availability. These metrics serve as observable variables that reflect the health and performance of services. An observable variable associated with each instance is used to infer dependency relationships within the infrastructure, as shown in Figure 2.2. This structured approach enables the framework to assess how deviations in one service propagate through the system, thus identifying critical points that may require redundancies to mitigate risks of cascading failures.

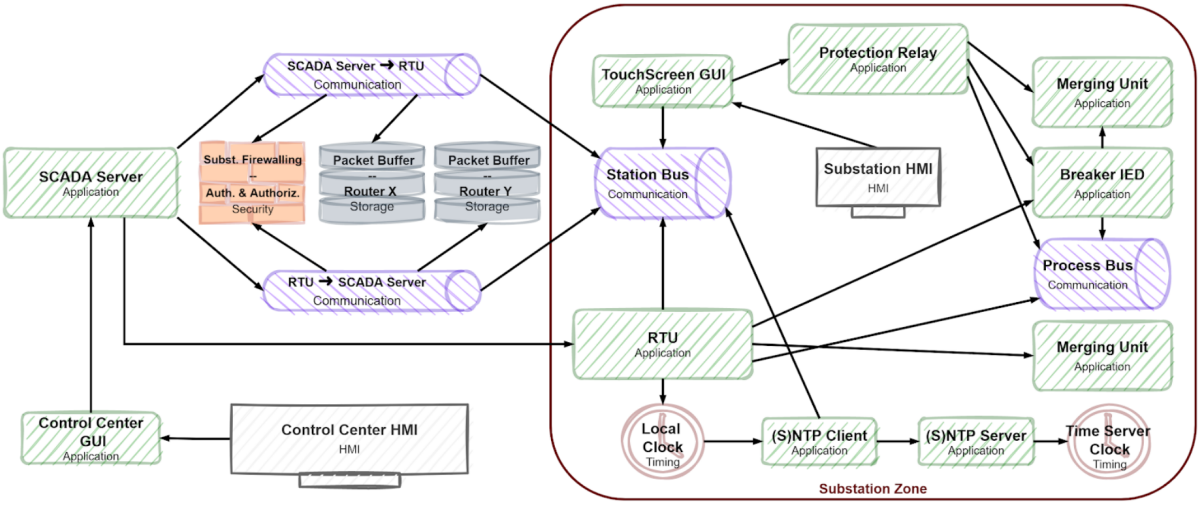


Figure 2.2: CSOD Dependency Graph of Cyber Services.

2.3.2 Inferring Dependencies with Causal Analysis

In CSOD, dependency inference is based on causal discovery techniques applied to time-series data gathered from monitoring tools such as `ping` and `traceroute`. These tools provide non-intrusive measurements, which are used to detect dependencies by observing temporal correlations between service metrics. The inference engine, detailed in the current project state, enables this causal analysis through algorithms like LPCMCI and tsFCI, which are adapted for time-series data with potential latent confounders that could obscure direct dependencies [1]. This inference is pivotal for the model as it uncovers the underlying structure of dependencies without requiring direct modifications to the system, thus preserving operational integrity while enabling real-time observability.

Each phase in the CSOD model's dependency inference process is designed to build a comprehensive view of the cyber-infrastructure's state:

- Phase 1: Structural Discovery. In this phase, dependency structure is approximated by observing correlations between time-series data from various service metrics. This phase may rely on tools like cross-correlation or Bayesian networks to identify candidate dependencies between services [2].
- Phase 2: Dependency Confirmation. This phase validates inferred relationships by testing the dependencies detected in Phase 1 against additional observational data and structural measurements, such as network topology or historical traffic logs. The goal is to confirm dependencies that have a high probability of impacting SLA metrics across service instances.

The CSOD framework's ability to automate the identification of dependencies and their effect on service adequacy provides a proactive approach to managing complex infrastructures. Moreover, this methodology aligns with cybersecurity practices by helping operators maintain system resilience through redundancy and targeted intervention at points identified as critical in the dependency graph.

2.3.3 Applications in Cyber-Physical Infrastructure

In critical infrastructure such as transmission systems, CSOD's dependency graph can predict how disruptions in one part of the network might lead to cascading failures. By simulating potential failure propagation scenarios, operators can identify which components would most benefit from added redundancy or enhanced monitoring. Figure 2.3 illustrates a simplified inter-instance dependency graph (IIDG) where dependencies between service instances are mapped out to facilitate targeted interventions.

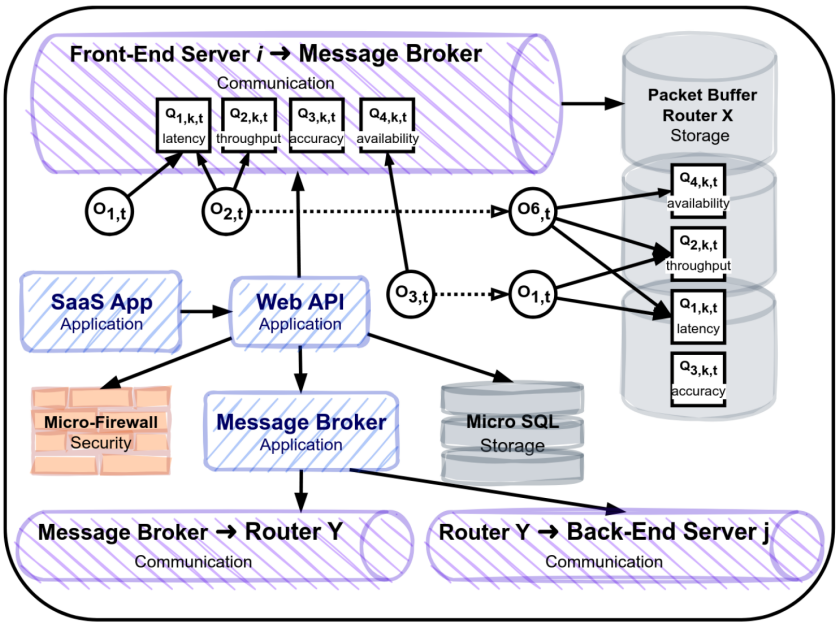


Figure 2.3: Simplified Inter-Instance Dependency Graph for SaaS Infrastructure.

The ability to identify and quantify service dependencies has numerous practical applications:

- Enhanced Resilience. By determining which service dependencies could lead to broad infrastructure disruptions, CSOD enables preemptive measures to bolster system resilience.
- Optimization of Maintenance Strategies. Dependency analysis informs maintenance schedules, as critical services identified in the IIDG are prioritized, potentially reducing downtime and service degradation risks.
- Risk Management. The CSOD framework provides the foundation for N-1 like criteria in cyber-infrastructure, enabling operators to anticipate service degradation impacts on dependent services and optimize redundancies accordingly.

In conclusion, the CSOD model represents a significant advancement in observability frameworks by offering a mathematically robust, service-oriented approach to dependency analysis. By establishing a dependency graph that highlights the relationships and risks inherent in complex cyber-infrastructures, CSOD aids operators in ensuring the resilience and reliability of critical digital services in cyber-physical systems.

2.4 SMOOTHIE (Loïc Champagne)

SMOOTHIE is a novel load-balancing solution designed specifically for congestion management within data center networks (DCNs). Developed using In-band Network Telemetry (INT) and Segment Routing over IPv6 (SRv6), SMOOTHIE leverages real-time network state monitoring to efficiently reroute traffic, thereby optimizing network performance and resource utilization. By employing a centralized controller, SMOOTHIE addresses limitations observed in traditional Equal-Cost Multi-Path (ECMP) routing and outperforms several other congestion-aware load balancers in dynamic environments.

2.4.1 Overview of SMOOTHIE's Architecture

The SMOOTHIE framework comprises three key components:

- Telemetry Data Collection: SMOOTHIE utilizes INT for gathering telemetry data directly from network packets, which provides insights into real-time network states, such as queue lengths, link utilization, and ingress/egress timestamps. This data is essential for detecting congestion points within the network. Notably, SMOOTHIE introduces the Proportional INT (p-INT) mechanism, enabling administrators to adjust the monitoring granularity by configuring the proportion of packets with telemetry headers. This feature helps in balancing between monitoring detail and network overhead, allowing prioritized observation of high-priority flows and minimized resource impact for non-critical traffic flows [3].
- Centralized Controller: At the core of SMOOTHIE's congestion management strategy is a centralized controller that processes telemetry reports to identify congestion and calculate optimal paths. Utilizing the "power of two choices"[4] approach, the controller assesses multiple paths and selects the least congested route for rerouting flows, thereby mitigating issues like route flapping, which can destabilize network performance.
- Flow Redirection via SRv6: SMOOTHIE applies SRv6 for directing traffic along calculated paths. As shown in Figure 2.4, SMOOTHIE adjusts paths dynamically when telemetry data indicates congestion, rerouting traffic through alternative nodes to avoid congested links. SRv6's programmable capabilities enable encoding of routing paths as segment lists, which are communicated to network switches, ensuring smooth traffic flow across the preferred route without modifying tenant virtual machines' network stack. This routing update is conducted over standard protocols such as gRPC, facilitating integration with existing network management tools.

2.4.2 SMOOTHIE's Contributions and Performance

SMOOTHIE brings several innovations to load balancing in DCNs:

- Dynamic Path Selection: Traditional ECMP methods rely on static path selection, often leading to inefficient traffic distribution. SMOOTHIE's dynamic path assignment is guided by real-time telemetry data, providing it the flexibility to respond to congestion proactively.
- Minimization of Route Flapping: A key challenge in many load-balancing algorithms is route flapping, which is the frequent changing of paths that disrupts network stability. As illustrated in Figure 2.5, SMOOTHIE

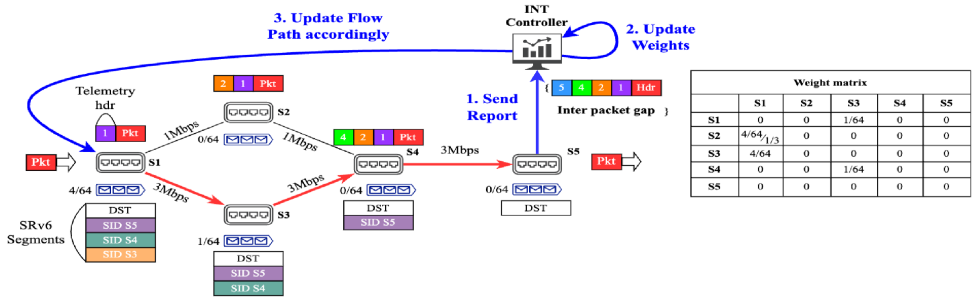


Figure 2.4: Routing evolution in SMOOTHIE. The initial path is [S1, S2, S4, S5] which is subsequently altered by the controller after processing telemetry data. The new route, encoded through SRv6, becomes [S1, S3, S4, S5] and is illustrated as the red path. Queue occupancy, denoted as x/y (where x represents the current occupancy and y is the queue size), is displayed below the switches.

limits route flapping by ensuring a maximum of two reroutes per flow, significantly reducing network latency and packet loss.

- Comparative Advantage Over ECMP and CLOVE [5]: Experimental results, depicted in Figures 2.6 and 2.5, show SMOOTHIE achieving superior load-balancing efficiency under high workload conditions. SMOOTHIE outperforms ECMP, especially in scenarios where traditional load balancers struggle with variable traffic patterns [3].

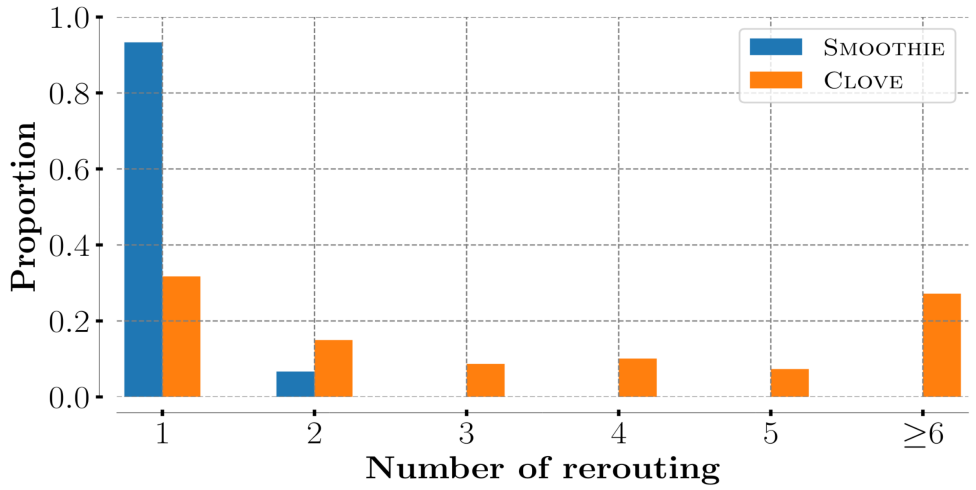


Figure 2.5: Assessment of route flapping prior to achieving convergence in SMOOTHIE and CLOVE. SMOOTHIE limits each flow to at most two reroutes, underscoring its optimized path selection and improved stability compared to CLOVE.

2.4.3 Implementation Details and Experimental Setup

The SMOOTHIE architecture is implemented using the P4 programming language, enabling a high degree of customization in packet processing. The experimental setup, based on a spine-leaf topology with 12 switches, utilized Mininet and P4utils for simulation. This topology connected 8 clients to 8 servers, allowing SMOOTHIE to demonstrate its effectiveness in a high-traffic environment that emulates typical DCN workloads. Flow completion time (FCT) was used as the primary performance metric, measured across various load levels to evaluate SMOOTHIE's response to different network conditions.

2.4.4 Implications for Cyber-Space Observability and Future Developments

SMOOTHIE's centralized and telemetry-informed approach provides critical insights into network health, aligning with the broader goal of cyber-space observability in hybrid infrastructures. By utilizing INT and SRv6, SMOOTHIE achieves high visibility into traffic patterns and congestion points, positioning itself as a key tool

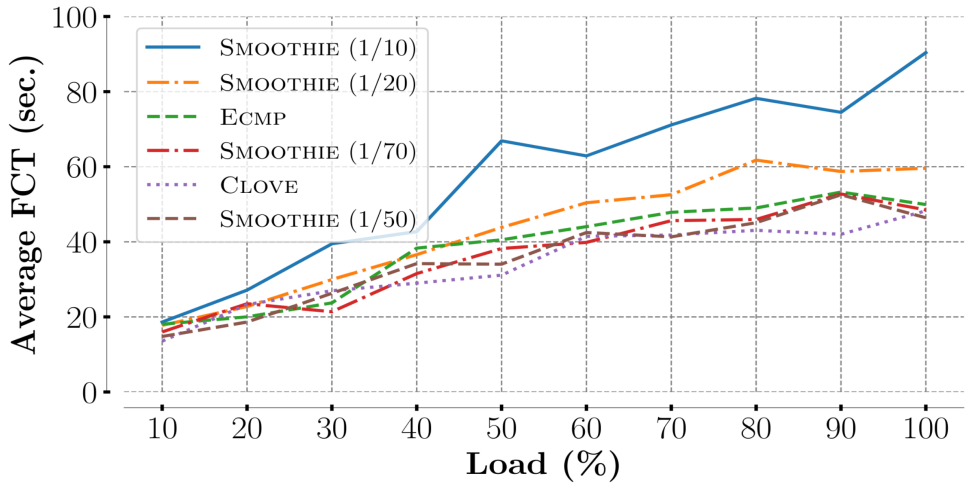


Figure 2.6: Comparison of SMOOTHIE with CLOVE and ECMP with respect to average flow completion time over different load. The threshold τ has been fixed to four RTTs. SMOOTHIE is tested with different p-INT values.

for DCN management. Future developments could explore further optimization of the p-INT parameters for different types of DCNs, potentially expanding SMOOTHIE's applicability to other network architectures outside of private domain settings such as WANs. Additionally, integration with automated configuration management systems could enhance its scalability in larger networks [3].

3

IT Anomaly Detection (Objective 2)

3.1 Introduction

The increasing interconnection between traditional power grids and digital infrastructures has transformed electrical systems into complex cyber-physical systems (CPSs). While this integration enhances operational efficiency and resilience, it also introduces significant vulnerabilities to cyber threats. In this context, IT anomaly detection plays a vital role in safeguarding the integrity of power system infrastructures. By identifying deviations from typical system behavior, IT anomaly detection aims to raise timely alarms about potential cyber intrusions, mitigating risks before they propagate into widespread system failures.

Objective 2 of the CYPRESS project addresses the development of anomaly detection techniques specifically tailored to the cybersecurity needs of power systems. IT anomaly detection in this domain focuses on recognizing suspicious patterns within the communication networks of cyber-physical power systems, particularly as they relate to critical grid operations. Unlike traditional detection methods, which often rely on power system measurements after a cyber-attack has been executed, early-stage anomaly detection leverages IT network-level analysis to identify cyber threats before their physical impacts become observable.

This objective builds on existing literature and methodologies in anomaly detection, including techniques from deep learning, signal processing, and encrypted traffic classification. The work presented in this chapter explores innovative approaches to detecting anomalies in encrypted communication flows, which have become increasingly prevalent due to the widespread adoption of security protocols like IPsec. The challenges posed by encrypted traffic—such as obfuscated malicious activities—are addressed through novel feature extraction and classification frameworks. These contributions aim to improve the timeliness and accuracy of anomaly detection within power system communication networks.

Furthermore, the anomaly detection frameworks developed here have potential applications beyond this objective. For instance, the concept of network polymorphism discussed in Objective 4 (Chapter 5) could leverage IT anomaly detection techniques as part of future developments to enhance adaptability and resilience against evolving cyber threats.

In summary, this chapter outlines the efforts undertaken to design and implement IT anomaly detection frameworks tailored to cyber-physical power systems. It provides an overview of the relevant literature, introduces the methodologies developed, and situates this work within the broader goals of the CYPRESS project.

3.2 Encrypted Traffic Classification for Early-Stage Anomaly Detection in Power Grid Communication Network (Tohid Behdadnia, Geert Deconinck, Can Ozkan, Dave Singelée)

3.2.1 Introduction

The impact of cyber-attacks on power grids is significant, causing a variety of consequences such as equipment impairment, load loss, and instability of the power system. In the most severe cases, persistent and sophisticated cyber-attacks can lead to widespread cascading failures and even result in blackouts [6]. It is therefore essential to implement anomaly detection techniques in cyber-physical power systems (CPPSs). In the realm of cyber security for power systems, anomaly detection algorithms commonly utilize measurements derived from power systems, which manifest when a cyber-attack is already effectively executed. A classic example of this is the utilization of false data detection techniques. The detection of false data is utilized as a means of mitigating the physical consequences of cyber-attacks [7]. This is while the majority of cyber-attacks, including those that targeted the Ukrainian power grid, highlight the crucial need for implementing early-stage anomaly detection as a defense mechanism in the cyber layer [8].

The implementation of encryption technology in power systems has enabled operators to achieve a notable improvement in the security and privacy of their infrastructure. However, the advantages of encryption are not exclusive to legitimate users, as threat actors have also leveraged this technology to evade detection and facilitate their malicious activities. This has resulted in a notable challenge in the detection of cyber-attacks in their early stages, as the proliferation of covert malicious behaviors poses a significant challenge to conventional communication network-based anomaly detection techniques [9]. To solve this problem, deep learning and signal processing techniques have been continuously introduced and applied for analyzing data packets and information flow to detect anomalies due to their potential to find unknown intrusions. For instance, in [10] the authors propose a flow-based classification method that can accurately characterize encrypted and VPN traffic using only time-related features. This approach reduces computational overhead by selecting a set of features that can be extracted with low computational complexity. In [11] the authors propose a deep learning-based approach for network traffic classification, which integrates feature extraction and classification into one system, called Deep Packet. The proposed framework can handle both traffic characterization and application identification, even for encrypted traffic. In [12], the proposed framework uses an LSTM network model for malicious traffic classification. This approach does not require the pre-processing of packets into flows, which leads to faster detection. In [13], Baldini explores the use of time-frequency analysis for encrypted traffic classification. The study applies different time-frequency transforms to features extracted from encrypted traffic and uses various machine-learning algorithms to classify the traffic. The approach is validated using a public dataset and shows superior performance in identifying different types of traffic. Tang et al. [14] present a novel representation learning method for encrypted traffic, which incorporates a diversity enhancement model that capitalizes on the diversity of images to represent the diversity of traffic samples. The proposed approach involves the transformation of encrypted traffic into Markov images and the utilization of a diversity maximization Markov GAN, based on the Simpson index, to generate new Markov images. The resulting balanced Markov image set is then subjected to classification via a CNN. The experimental results demonstrate the efficacy of the proposed method in predicting the entire dataset space with a limited number of original samples.

Drawing upon prior work on encrypted traffic classification, we presents a framework for early-stage anomaly detection in CPPSs. The proposed method employs wavelet transformation and byte-to-image transformation for feature extraction from the network traffic signal (NTS) and packets payload, respectively. To the best of our knowledge, our study is among the first to explore the detection of malicious traffic at the packet level in the context of CPPSs. The unique characteristics of power grid communication networks pose challenges to anomaly detection in this domain, as NTSs in power systems are distinct from those in other communication networks and are often not publicly available for studies. Our work addresses this challenge by creating realistic data flows that simulate the NTSs in CPPSs. Furthermore, our study employs internet protocol security (IPsec) encryption to secure network traffic. Compared to transport layer security (TLS) encryption, IPsec is implemented at the network layer (layer 3) of the open systems interconnection (OSI) model and encrypts the entire IP packet, including the header and payload. This encapsulation of the payload at the network layer

makes it more challenging to inspect and classify the traffic contents.

3.2.2 Data Flow in Power Systems: An Overview of Data Flow Types and Transmission Protocols

As power system operations become more complex and interconnected, the need for secure and robust communication networks is becoming increasingly vital. Consequently, a more comprehensive understanding of information flow is necessary. This section aims to provide an overview of various types of data flows between substations and the control center, including periodic, random, and burst flows, as well as the protocols employed for transmitting this data.

Periodic Data Flow

Periodic data transmission is a critical component of power systems, as it facilitates the exchange of diverse information types between substations and control centers. Telemetry data, SCADA data, power quality data, and load data are among the various forms of information transmitted periodically. Telemetry data is transmitted every few seconds, whereas SCADA data is delivered in real-time. Power quality data and load data are typically conveyed at periodic intervals, such as every minute or every fifteen minutes, respectively. These data flows are essential for monitoring, control, fault detection, diagnosis, and system optimization. Communications network protocols, such as DNP3, IEC 61850, Modbus, and IEEE C37.118, are employed to ensure reliable, secure, and efficient data transmission.

Random Data Flow

In the realm of power systems, various types of random data flows occur between substations and control centers. These data flows include event logs, fault records, maintenance data, weather data, and market data. The transmission protocols used for these data flows depend on the type of data being transmitted. For event logs, common protocols used include DNP3, IEC 61850, Modbus, and OPC. Fault records are typically transmitted using the IEC 61850 protocol, while maintenance data is typically transmitted using proprietary protocols or DNP3, IEC 61850, and Modbus. Weather data can be transmitted using various protocols such as AWIPS, MODIS, and NEXRAD. Market data is typically transmitted using specialized protocols such as FIX and FpML.

Burst Data Flow

Burst data flow refers to the transmission of large amounts of data in short bursts or packets. This type of data flow is essential for real-time monitoring, control, and protection of the power system. Protocols used for transmitting burst data flows are typically designed to ensure fast and reliable communication with minimal latency and high data throughput. Commonly used protocols for sending burst data flows between substations and control centers include phasor measurement unit (PMU) data, protection relay data, control commands, and video data. For PMU data, the most commonly used protocol is the IEEE C37.118 protocol. Other protocols used for transmitting PMU data include the IEC 61850-90-5 protocol and the IEEE C37.118.1a extension. For protection relay data, the most commonly used protocol is the IEC 61850-9-2LE protocol. Another protocol used for transmitting protection relay data is the IEC 60870-5-103 protocol. For control commands, the most commonly used protocol is the IEC 61850 protocol. Other protocols used for transmitting control commands include the DNP3 protocol and the Modbus protocol. For video data, the most commonly used protocols are IP-based protocols such as RTP and RTSP.

The selection of a protocol for transmitting all types of data flows is dependent on various factors such as the requirements of the specific power system application, the frequency and type of data reporting, and the communication infrastructure used.

3.2.3 Securing Information Flow in Power Systems: The Use of IPsec and Consequential Cyber Threats

To secure the information flow between power system substations and control centers, there are several options, including using virtual private networks (VPNs), secure sockets layer (SSL), transport layer security (TLS), secure shell (SSH), and IPsec. IPsec is a popular choice due to its end-to-end encryption and authentication at the network layer. It provides flexibility in terms of security policies, supports various encryption algorithms

and key exchange methods, and can be easily implemented on most network devices. While IPsec provides a secure communication channel between two endpoints, it does not protect against all types of attacks.

Attackers can take advantage of the encrypted domain provided by IPsec by using encrypted malware that is disguised as legitimate traffic to evade detection by security measures. Because the traffic is encrypted, it may be difficult to detect the presence of malware. Additionally, attackers can use covert channels to hide their malicious behavior within legitimate traffic, such as using unused fields in legitimate packets to hide their data. Advanced persistent threats (APTs) are long-term, targeted attacks that are designed to evade detection by security measures. APTs often use advanced techniques, such as encrypted command and control channels, to remain hidden within the network. Insiders, such as employees or contractors, who have legitimate access to the network can use encrypted communication channels to exfiltrate sensitive data or launch attacks from within the network.

To mitigate these types of attacks, it is important to implement additional security measures such as communication-based anomaly detection algorithms, network traffic analysis tools, and data loss prevention solutions. These measures can help detect and prevent attacks that may be hidden within encrypted traffic.

3.2.4 Proposed Methodology for Detecting Malicious Activities

Our proposed approach involves the capture and processing of data packets from the power systems communications network to identify malicious activities. Specifically, we perform a byte-to-grayscale transformation on the byte information of each packet, resulting in grayscale values between 0 and 255. These values are then used to populate an N×N matrix, which is further partitioned into two sub-matrices. The first sub-matrix consists of the grayscale values obtained from the byte-to-grayscale transformation of data packets, while the second sub-matrix contains wavelet coefficients, also in grayscale, extracted from the NTS at the time of packet capture. This partitioned matrix serves as the input to a convolutional neural network (CNN) trained to detect malicious packets. Figure 3.1 shows the graphical overview of the proposed wavelet-based feature extraction and CNN-based packet classification method.

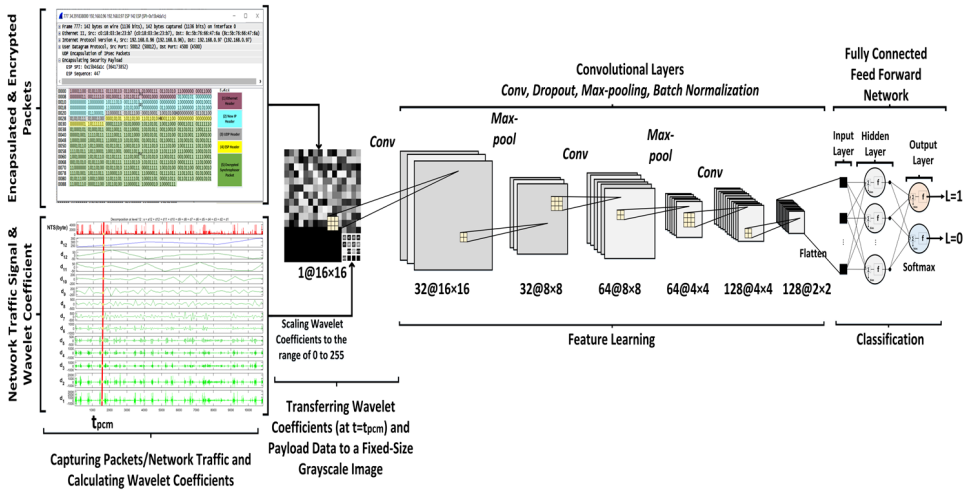


Figure 3.1: Graphical overview of the proposed wavelet-based feature extraction and CNN-based packet classification.

3.2.5 Conclusion

The proposed framework for early-stage anomaly detection using deep-learning-based traffic classification presents a promising solution for securing communication networks in CPPSs. The use of wavelet transforms and byte-to-image transformation for feature extraction from encrypted traffic flow in the networks secured by the IPsec/ESP protocol suite results in a highly accurate method for detecting abnormal traffic, achieving 93% accuracy in our experiments. This approach represents a significant improvement over existing methods that rely on power system measurement anomalies that appear only after an attack has been executed, as it enables early-stage detection of attacks through communication network-based anomalies. One potential avenue for future research could involve exploring the efficacy of the proposed method for identifying additional types of malicious packets beyond those specifically utilized in port scanning activities. Moreover, accounting

for other communication protocols during the data flow generation process would result in more realistic data flows, consequently enhancing the reliability of the detection algorithm for deployment in actual systems.

4

Cyber-Operator Characterization (Objective 3)

4.1 Introduction

The growing interdependence between electrical power grids and information systems has led to the evolution of modern power systems into complex cyber-physical systems (CPS). In this context, the effective operation and resilience of such systems are critically influenced by the actions and decisions of operators managing both the cyber and physical domains. Objective 3 of the CYPRESS project, "Cyber-Operator Characterization," addresses this intersection by focusing on understanding the role of cyber-operators in ensuring the security and stability of these interconnected systems.

Objective 3 aims to define the state space of cyber-infrastructure and investigate how it can be influenced through operator actions. Specifically, this involves identifying the set of preventive and corrective actions available to cyber-operators, as well as understanding the factors influencing their decision-making processes. These insights are expected to enhance the operational framework of cyber-physical systems by improving risk management and response mechanisms against cascading outages and other systemic threats.

Within this objective, the primary focus lies on a single workstream—Cyber-Operator Characterization—which thoroughly examines manual corrective actions (MCAs) performed by operators in response to contingencies. These actions, inherently uncertain and prone to imperfections, play a significant role in the progression or mitigation of cascading outages. By systematically analyzing the probabilistic imperfections of MCAs and integrating them into risk assessment models, this work contributes to a more comprehensive understanding of how operator behavior influences the resilience of power systems.

The outcomes of this effort are pivotal for enhancing the security and operability of cyber-physical power systems. By bridging gaps in existing security management frameworks, the research provides actionable insights into the development of tools and methods for mitigating risks associated with operator errors. This aligns with the broader objectives of the CYPRESS project, which seeks to establish innovative knowledge and methodolo-

gies for addressing cyber-physical risks in the bulk electric energy supply system, thereby ensuring reliable and secure grid operations in the face of evolving cyber and physical challenges.

4.2 Cyber-Operator Characterisation (Shahab Kamyab, Pierre Henneaux, Pierre-Etienne Labeau)

4.2.1 Introduction

Cascading outages occur due to interdependent failures of components in the electric power network, happening successively over a short period due to the lack of timely corrective actions following initial contingencies. They are recognized as key contributors to significant disturbances and blackouts in transmission grids. Analyzing past large-scale blackouts and system disturbances highlights the substantial role of delayed, incomplete, or failed corrective measures in escalating cascading outages [15].

Moreover, the security management and risk assessment of power systems are increasingly strained by limited investments in new infrastructure, rising loads, and the growing penetration of renewable energy sources and distributed generation. Consequently, operator failures have become particularly critical in modern power systems, which operate under tight security margins and face increased variability from renewable generation and complex interdependencies. These challenges have significantly heightened the reliance on effective corrective actions as a vital mechanism for maintaining system stability following contingencies [16].

While it is ideal for operators to make optimal decisions, corrective actions—especially those performed manually—are inherently uncertain and prone to imperfections [17]. Despite their importance, the probabilistic nature of these actions has received limited attention in conventional security and risk assessments of cascading outages. Only a few studies have addressed the impact of manual operator performance imperfections in probabilistic risk assessments of cascading outages in electrical power systems, underscoring a critical gap in the literature [18]. Neglecting the uncertainties in manual corrective actions (MCAs) may lead to an underestimation of system risks, thereby undermining the effectiveness of security management strategies [17].

4.2.2 Methodology

Integrating the probabilistic imperfections of MCAs into security assessment and management frameworks is essential to address these challenges. This requires a systematic approach involving the following steps:

1. Characterizing MCA Behavior: Accurately identify and describe all possible failure modes of MCAs, ensuring a comprehensive understanding of their behavior.
2. Assigning Probabilities: Allocate justifiable probability values to each identified failure mode.
3. Evaluating Failures: Assess the likelihood and impact of manual action failures under various operational scenarios.
4. Analyzing Cascading Effects: Investigate how imperfections in corrective measures influence post-contingency system states and contribute to cascading outages.
5. Updating Risk Models: Enhance probabilistic risk assessment models by incorporating the quantified probabilities of MCA imperfections.

As shown in Figure 4.1, the Human Reliability Analysis Optimal Power Flow (HRA-OPF) framework, initially developed by the authors of [19] and later extended in [20], addresses these requirements. The HRA-OPF framework quantifies the contribution of manual errors during post-contingency corrective actions. It first estimates the failure probability of manual operators based on the time margin ratio.

Corrective actions are defined for each insecure contingency by solving an Optimal Power Flow (OPF) problem. The objective is to minimize the cost of corrective actions or the required time to perform them. The time margin ratio is calculated as the ratio of available time after contingency to the required time for implementing corrective actions. Available time is the interval between the occurrence of an insecure contingency and the next triggering event, while required time represents the duration needed for a typical power system operator to diagnose the situation and implement corrective actions. Corrective actions include redispatch of the output power of generators or load-shedding.

The study evaluates and compares the risk values for three scenarios: (1) perfect (error-free) MCAs, (2) no MCAs, and (3) imperfect MCAs. The detailed methodology and results are elaborated in the report of D2.3, which focuses on advanced risk assessment methodologies.

4.2.3 Results and Discussion

The HRA-OPF framework was applied to the New England Test System (NETS) grid to evaluate the impact of manual corrective actions (MCAs) on cascading outage risks. The results indicate that imperfections in MCAs significantly contribute to the risk of cascading outages. As illustrated in Figure 4.2, completely neglecting MCAs dramatically increases the Complementary Cumulative Disturbance Probability (CCDP) of load shedding in cascading outage scenarios. However, even imperfect MCAs were shown to augment risk compared to ideal (error-free) scenarios.

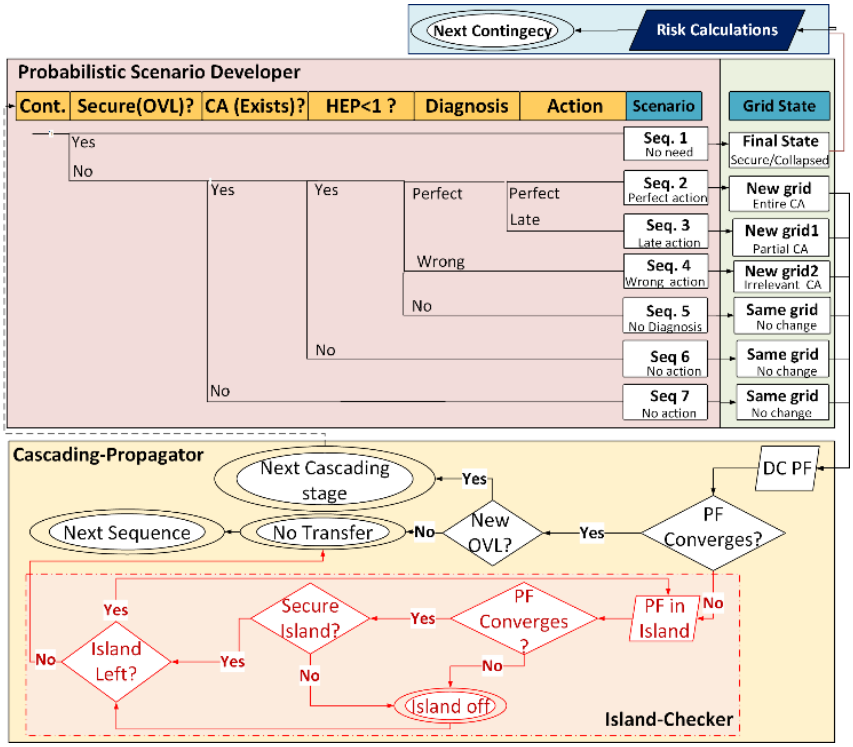


Figure 4.1: Flowchart of HRA-OPF
(Conting: Contingency, OVL: Overloading, HEP: Human Error Probability, PF: Power Flow)

The modern power system relies heavily on information infrastructure for effective operation. Without this infrastructure, even the most visible components of the grid—such as transmission lines, substations, and generating plants—cannot function efficiently. Similarly, operators' ability to manage and mitigate outages depends heavily on their situational awareness and access to critical system information [21].

Despite its advantages, the smart grid is highly vulnerable to security threats due to inherent weaknesses in communication technology [22]. As a result, cybersecurity has become a critical aspect of power system security, complementing physical security measures. Cyber-attacks can severely disrupt the control systems underpinning electric power grids, with potentially catastrophic consequences.

State estimation (SE) is a vital control center function that calculates bus states using data from the Supervisory Control and Data Acquisition (SCADA) system. The results of SE are fundamental to numerous power system applications, such as economic dispatch and contingency analysis. However, a successful false data injection attack (FDIA) can compromise measurements from grid sensors, introducing undetected errors into the estimates of state variables, including bus voltage angles and magnitudes [23]. These falsified SE results can mislead operators, causing incorrect diagnoses (wrong diagnosis) or neglect of ongoing issues (no diagnosis). Such diagnostic failures may result in erroneous or missed actions, leading to significant physical or economic impacts on power systems. Consequently, robust defensive measures must be designed to protect the grid

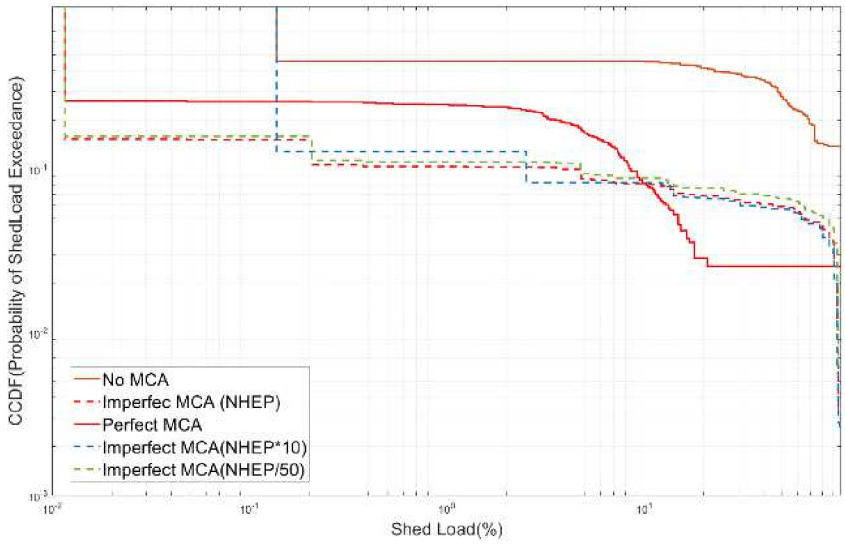


Figure 4.2: CCDF[1] of loss of load percentage for three cases
(case 1: No MCA, Case 2: Perfect MCA, case 3: Imperfect MCA with Nominal HEP (NHEP), NHEP*10, NHEP/10)

from these threats.

Addressing these challenges aligns with current efforts to bridge the gap between cyber and physical system vulnerabilities. Future studies will extend the HRA-OPF framework in [19] to model and assess the uncertainties associated with MCAs under cyber-attacks. This approach aims to enhance the accuracy of risk assessments in cyber-physical power systems by considering how cyber-attacks impact the precision of MCAs and the resulting risk of cascading outages. Future research will focus on two critical axes[2]:

1. The Influence of Cyber-Attacks on MCA: Investigating their contribution to the risk assessment of cascading outages.
2. Effectiveness of HRA Methodologies: Studying their feasibility in preventing and mitigating cyber-attacks during incident handling [24][3].

4.2.4 Conclusion

This report highlights the contribution of imperfect manual corrective actions to the risk of cascading outages in modern power systems. For this, the failure probability of grid operators upon non-secure contingencies is estimated using Human Reliability Analysis (HRA). Accordingly, the developed HRA-OPF framework provides a more accurate assessment of cascading outage risks by incorporating the probabilistic nature of operator errors during manual corrective actions.

In the context of cyber-physical power systems (CPPS), cyber-attacks, such as False Data Injection Attacks (FDIAs), further emphasize the importance of this approach. FDIAs can compromise state estimation outputs, mislead operator decision-making, and significantly increase the risk of cascading failures.

Future work will focus on extending the HRA-OPF framework along two critical axes:

1. Evaluating the Impact of FDIA on SE: To assess how cyber-attacks affect state estimation precision and cascading outage risk.
2. Exploring HRA Techniques for Cyber Threat Mitigation: To study the feasibility of using HRA methodologies for preventing and mitigating cyber threats during incident handling.

[1] The Complementary Cumulative Distribution Function (CCDF), also known as the survival function, is a statistical function that represents the probability that a random variable X takes on a value greater than or equal to a specific value x. It is complementary to the Cumulative Distribution Function (CDF). Using this CCDP gives the probability of having a load shed percentage higher than a value.

[2] The details of the framework will be elaborated in D2.3, as it aligns more closely with the context of risk assessment in cyber-physical power systems (CPPS). The focus of that report will be on the advanced methodologies for evaluating risks in CPPS.

[3] The Incident Handling Process outlined by SANS (SysAdmin, Audit, Network, and Security) provides a structured framework for systematically managing and mitigating the impact of cyber incidents. Widely used in cybersecurity operations, its integration into power system security assessments can enhance operator preparedness for cyber-induced disruptions, thereby minimizing the risk of cascading outages caused by delayed or improper responses.

This integrated approach bridges the gap between human reliability and cybersecurity, enhancing the security and resilience of real-time operations in cyber-physical power systems.

5

Design of Cyber-Attack Resilient Cyber-Physical Systems (Objective 4)

5.1 Introduction

The integration of information and communication technology (ICT) within the bulk Electric Energy Supply (EES) system marks a transformative shift towards cyber-physical systems (CPS). While this advancement enables smarter and more efficient power systems, it also introduces unique vulnerabilities to cyber-attacks, posing significant challenges to system resilience. Objective 4 of the CYPRESS project addresses these challenges by focusing on the design of cyber-attack resilient CPS, ensuring robust performance even under partial system compromise. Unlike prior objectives that emphasize diagnostics, anomaly detection, and operator actions, this objective aims to maintain nominal behavior within the cyber-infrastructure itself, safeguarding its ability to provide critical services without disruption to the power system.

Building on the foundation laid by earlier objectives and complementary to past research on ICT-aware power system operations, Objective 4 seeks to enhance the robustness of cyber-infrastructures through innovative designs, algorithms, and protocols. These efforts aim to create systems capable of withstanding cyber-threats while preserving their operational integrity, even in adverse conditions. This objective aligns with the broader goal of Work Package 3 (WP3) to mitigate cyber-physical security risks and benefits from insights provided by Work Package 1 (WP1), such as reliability indicators, relevant components, and benchmarks.

The following sections will explore various research efforts under Objective 4, including decentralized group authentication protocols, system polymorphism techniques, and solutions for addressing supply chain insecurities. These works collectively contribute to a resilient cyber-physical landscape, ensuring security and reliability amidst evolving cyber threats.

5.2 Decentralized Group Authentication with Membership Verification in Islanded Smart Grids (Wilson Daubry, Jean-Michel Dricot, Pierre Henneaux)

5.2.1 Introduction

The electrical grid is a crucial aspect of our society, providing power to homes and businesses on a daily basis. One of the main goals of grid operators is to match electricity generation to consumption in real-time and to ensure a stable power supply, thereby preventing blackouts. There are two types of grid operators: transmission system operators and distribution system operators. Transmission system operators handle high voltage lines[1] and facilitate the transport of electricity from generation to consumption areas, while distribution operators distribute electricity from the transmission grid to local areas for consumption. Large industries are often directly connected to the transmission system operator [25].

Islanding in power grids occurs when a portion of the grid becomes disconnected from the main grid. Historically, this could have led to a blackout in that region due to the physical separation of producers and consumers by many kilometers. However, the increasing decentralization of energy generation from sources such as distributed renewable energies, plug-in hybrid vehicles, and prosumers allow microgrids [26] to continue functioning independently [27].

Power grids are usually interconnected with a communication network at both the transmission and distribution levels. At transmission level, the communication network is used to monitor the power flow and ensure that the energy produced is effectively distributed to the designated consumption areas. This is achieved by the transmission of consumption and generation measures at different points of the grid. On distribution level, the communication network serves as the backbone for the monitoring and control of smart grids, facilitated by the widespread use of smart meters. A smart meter is a device that enables real-time monitoring of electricity consumption or generation at each node of the network. A comprehensive illustration of these nodes will be presented in detail in Section 5.2.2. The collected data might be transmitted to a control center for billing purposes, taking into account the real-time price of energy and to maintain a balanced generation and load on the grid. Furthermore, smart grids should allow control centers to send instructions to the smart meters.

An analogy can be drawn between the islanding of a portion of the power grid and a similar occurrence in terms of telecommunications devices. As previously mentioned, smart grids are composed by two distinct networks: a power network and a telecommunication network. The issue of islanding in the power network has been extensively researched and solutions have been developed for detection [28] and operation [29] under islanding conditions. However, there appears to be limited research on the topic of islanding in the telecommunication network. This study aims to fill this gap by introducing a novel authentication method for verifying the authenticity of the nodes in the network.

The islanding scenario discussed in this paper has several notable features. Firstly, it isolates the microgrid from the central authority or trusted third party. Secondly, it supposes a match between the energy consumption and generation. However balance can result from preventive or corrective actions to adjust the consumption and the generation. Not having an equilibrium at a point in time does not mean that it can not be reach later. Thirdly, the microgrid is equipped with a widespread deployment of smart meters. The occurrence of islanding may be attributed to various factors such as earthquakes, fires, or acts of terrorism.

This paper presents a modified form of the group authentication scheme outlined in [30] to offer a robust solution for the authentication of nodes in a telecommunications system. The primary objective of the authentication is to ensure that only genuine group members are recognized and accepted by other members. Thus, the scheme must guarantee that forging a valid share is infeasible, as established in Harn's algorithm described in [31]. Furthermore, the authentication scheme should thwart malicious nodes from participating in the authentication process, either all group members are authenticated together or none are. This scheme enables the authentication of legitimate devices while detecting any malicious attempt to interfere with the authentication process. Although the proposed scheme has direct implications for the electrical grid, it can also be applied to any Internet of Things system.

This paper is organized as follows: Section 5.2.2 provides an overview of the necessary background information for the proposed protocol. Section 5.2.3 presents the proposed protocol in details. Section 5.2.4 examines the

[1] Depending on countries, but mostly starting from around $30kV$

protocol and its potential implications. Finally, Section 5.2.5 offers a concise conclusion and considers the future direction of research in this area.

5.2.2 Preliminaries

In this section, we present a system model of the microgrid under consideration and define the terms linking the power grid with its telecommunication component. The relevant concepts introduced include secret sharing, Nyberg's one-way accumulator for one-way hash functions, group authentication, and a lightweight version of verifiable secret sharing. These concepts are leveraged to develop a new group authentication protocol with the capability to detect intruders.

Structure of the microgrid network

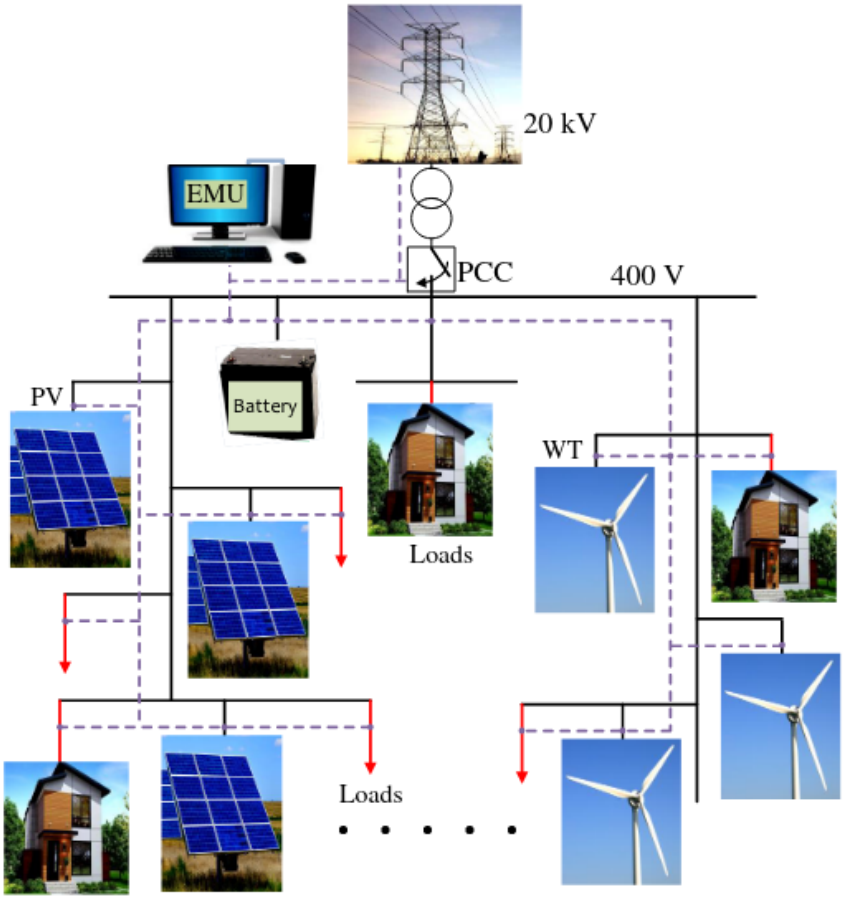


Figure 5.1: A community microgrid [32]

As mentioned in the introduction, the power grid can be divided into two main components: the transmission level and the distribution level, which are usually managed by separate operators. At the distribution level, a microgrid is depicted in Figure 5.1. Smart meters are connected to various loads and energy generation units, enabling the monitoring of the entire microgrid's load. All this information is transmitted through the network to the control center, where operators can balance the grid's load.

Smart meters play a crucial role in smart grids [33], serving as the bridge between the power grid and the telecommunication network by generating, transmitting, and receiving data. Throughout this paper, terms such as node, group member, or participant in a group authentication or secret sharing process will refer to these smart meters.

Shamir's Secret Sharing

The goal of a secret sharing scheme is to divide a secret among a group of n participants, so that if a minimum of m members work together, the secret can be recovered. It is critical that the number of group members, n, exceeds the predetermined threshold value, t, for the reconstruction to be successful. If this threshold is not met, no information about the secret will be disclosed. These conditions can be succinctly stated as $t \leq m \leq n$.

The scheme consists of two main phases. In the first phase, a trusted third party generates a secret s, and selects a random polynomial of degree $t \leq n$, taking the following form:

$$f(x) = a_0 + a_1 x + ... + a_{t-1} x^{t-1} \quad (mod p) \tag{5.1}$$

Where $s = f(0) = a_0$ is the secret and the degree t of the polynomial represents the minimum number of members required to perform a secret reconstruction. All coefficients $a_i, i \in \{1, ..., t-1\} \in GF(p)$ with $p > s$ where $GF(p)$ is a finite field[2]. The trusted third party then needs to compute $s_i = f(x_i)$ for each group member, with x_i being the public information belonging to each group member, while s_i is the secret share of the secret associated with member M_i.

Secondly, when a group authentication is performed, each member broadcasts its share s_i. Using Lagrange interpolation, each group member should be able to reconstruct the initial secret :

$$s = f(0) = \sum_{i=1}^{t} s_i \prod_{r=1, r \neq i}^{t} \frac{-x_r}{x_i - x_r} \quad (mod p) \tag{5.2}$$

It is important to understand that (i) the reconstructed secret is exactly the same as the original one and not an approximation of it and (ii) that without enough member to reconstruct the secret, no information on the secret s can be computed.

Nyberg's one-way accumulator for one-way hash function

A hash function is a one-way transformation that maps an input to a fixed-length output with an extermely low probability of collision[3], even when the size of the output space is smaller than that of the input space. Hash functions are not easily reversible, they are a surjective function and finding a collision requires trial and error.

The Nyberg's one-way accumulator for one-way hash function, noted as hash function in this paper, as described in [34], [30] needs to have several properties. First, it needs to be quasi-commutative: $H(H(\alpha, \beta), \gamma) = H(H(\alpha, \gamma), \beta)$, where H is the hash function. Second, it needs to be absorbent, such that $H(H(\alpha, \beta), \beta) = H(\alpha, \beta)$. Third, a value α already in an accumulated value V should result in $H(V, \alpha) = \alpha$, which is a corollary of the second property.

Group Authentication

The Shamir's secret sharing scheme presented above allows for the retrieval of a secret between different member of a group, each member having a share of this secret. This original purpose scheme was not design to perform a group authentication but rather share a secret across a certain amount of group members [35]. The concept of group authentication was introduced by Harn [31], and its objective is to perform many-to-many authentication based on Shamir's secret sharing. To do so, one key feature needed was the possibility to perform an asynchronous secret reconstruction.

Since the secret s was the independent term of the polynomial, retrieving it allows for a total reconstruction of the polynomial function $f(x)$ used to compute the shares. Hence, the forgery of a share is possible for an individual that did not belong to the group at first.

In the original proposal, this feature was unfeasible, as the polynomial $f(x)$ could be retrieved once t shares were broadcasted, making it possible for malicious members to forge new shares and blend in. To prevent this, the share $s_i = f(x_i)$ is obfuscated, such that the secret is not based on a single polynomial, but rather on a linear combination of several polynomials.

The trusted third party selects k random polynomials $f_l(x), l = 1, 2..., k$ of degree $t-1$, where $kt > n-1$. k tokens are generated for each group member M_i : $f_l(x_i), l = 1, 2, ..., k$. For a secret s, the trusted third party selects integers $w_j, d_j, j = 1, 2, ..., k$ in $GF(p)$ such that :

[2] Also called a Galois Field

[3] When two input points towards the same output.

$$s = \sum_{j=1}^{k} d_j f_j(w_j), \tag{5.3}$$

where $w_i \neq w_j$ for every pair of i and j. Each $w_j, d_j, j = 1, 2, ..., k$ and $H(s)$ are public information, where H is a hash function[4].

The group authentication is then performed as follows. Each member that wants to participate in the group authentication computes the following value, which is called a Lagrange component:

$$c_i = \sum_{j=1}^{k} d_j f_j(x_i) \prod_{r=1, r\neq i}^{m} \frac{w_j - x_r}{x_i - x_r} \quad (mod p). \tag{5.4}$$

The notation c_i is used to make the difference with the notation s_i used in subsection 5.2.2, but the idea remains the same, it is the private share that will later be used in the group authentication. Then, after receiving the c_i for other members, one can reconstruct the secret using the following formula:

$$s' = \sum_{i=1}^{m} c_i \quad (mod p). \tag{5.5}$$

The value $H(s')$ can be compared with the publicly known $H(s)$ to validate that the group authentication worked. Here, s' is the secret reconstructed based on the accumulation of the different shares, while s is the original secret generated by the thrusted third party. In this case, later authentication of a group member M_h is still possible, since it is impossible to forge a share based on the knowledge of other shares and the secret s is hidden behind a hash function and only known as $H(s)$.

Ligthweight Verifiable Secret Sharing

Verifiable secret sharing was first proposed by Stadler in [36]. The objective of verifiable secret sharing is to prevent a malicious node from participating in authentication by providing means of verification for the shares that are broadcasted by the different nodes. This verification can be performed by any node on the network, and being a real member of the group is not a requirement.

In [30], a lightweight version of the verifiable secret sharing scheme is proposed. That paper presents a modified[5] version of the verifiable secret sharing algorithm, which is divided into two main parts: (I) Share Generation and (II) Secret Reconstruction.

Share Generation First, the trusted third party chooses a polynomial $f(x) = a_0 + a_1 x + ... + a_{t-1} x^{t-1}$ with a_0 being the secret. The coefficients $a_i \in \mathbf{Z}_q^*$. Shares corresponding to $s_i = f(x_i)$ are distributed to each node P_i, where x_i is the unique identifier of the node. The verification data is then computed based on the hash function as follows:

$$V = H(...H(H(k, s), s_1), ...s_n). \tag{5.6}$$

The trusted third party then provides each node P_i with its share s_i, the verification data V, and the function H.

Second, after receiving its share s_i every participant P_i will check the authenticity and integrity of the share. The participant will perform the following verification: $H(V, s_j) = V$. If the verification does not hold, the first step needs to be done again.

Secret Reconstruction Third the phase begin when each participant P_i releases its share s_i, and the combiner will then validate the validity of each share by computing $H(V, s_i) = V$.

Fourth, the secret is reconstructed by the combiner which reconstructs the secret s based on the received shares $s_1, s_2, ...$ Finally, the combiner validates the reconstruction using $H(V, s) = V$.

This scheme is based on the notion of accumulator [34]. The trusted third party produces a hash function. Each member of the group receives their own share and a verification data V, which allows them to verify that their

[4] In this particular case, the hash function is not a Nyberg's one-way accumulator for one-way hash function. This will only be the case when the hash function possesses two arguments.

[5] To ensure that devices with a low computational power can perform the needed computations.

own share is valid. This verification data also allows the group member to verify that the shares provided by the other group members are valid.

5.2.3 Proposed approached

The proposed protocol is divided into three steps: a preparation phase, an islanding phase and a malicious user identification phase.

1. The first step is a preparation phase, which must take place while the network is fully operational. This step must be completed before an islanding event occurs. Only the trusted third party can add new nodes to the telecommunication network by providing them with a share of a global secret.
2. The second step is a recovery phase that occurs whenever an islanding of the microgrid occurs. In this recovery phase, nodes exchange a manipulation of their share to retrieve the global secret together.
3. During the optional third phase, possible intruders are detected and rejected of the group authentication.

The trusted third party is considered as the central authority that usually performs the authentication of the devices in the network. It is similar to a certificate authority in the Internet. In normal operation, the connection between the trusted third party and the devices is considered secure. Each node, including smart meters for consumers, smart meters located at generation units (such as wind or solar), and plug-in hybrid vehicles, will be considered as a member of the group.

Preparation phase

The preparation phase, as mentioned earlier, takes place while the network is in normal operation. The trusted third party selects k random polynomials $f_l(x), l \in \{1...k\}$ of degree $t-1$ with n being the number of members in the group such that $(kt > n-1)$. With t the threshold of the Shamir's secret sharing scheme, n the number of nodes. The trusted third party generates k tokens in the form of $f_l(x_i)$ for each element of the grid M_i. For any secret s, the trusted third party finds integer $w_j, d_j, j = 1, 2...k$ in $GF(p)$ such as in the equation 5.3 of Section 5.2.2.

Where $w_i \neq w_j$ for every pair of i and j. The trusted third party then computes the shares of the different group members based on the equation 5.4 of Section 5.2.2. The trusted third party selects a secret key z to compute the verification data V using a hash function H based on equation 5.6 of Section 5.2.2.

The trusted third party will subsequently distribute the required materials for phases 2 and 3. First, the unique value c_i for each node M_i will be transmitted via an encrypted and authenticated channel. Next, the accumulated value V and the obfuscated secret $H(s)$ will be disseminated to every node in the network. As this information is public, a secure channel is not necessary.

Islanding phase

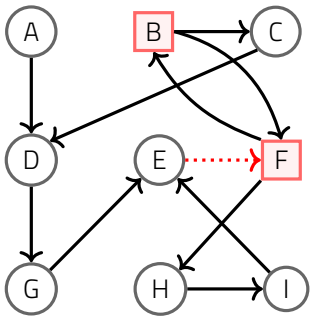
In the event of a disruption that breaks the connection between the trusted third party and the microgrid or elements of the grid, the usual authentication method is no longer possible. Therefore, the restoration phase needs to begin. Each node M_i willing to join the group authentication will release its share c_i, which was computed by the trusted third party during the preparation phase. Upon receiving the shares of the other members, each member can then try to reconstruct the value s using equation 5.5 of Section 5.2.2

The reconstructed secret s' needs to be compared to the Hash of the real secret propagated by the trusted third party. If $H(s) = H(s')$, then every member that shared its share and was used in the computation of s' is authenticated. It is important to note that this scheme is asynchronous, meaning that any member can take part in the group authentication at different times. However, a minimum number of members is required for the authentication to take place.

If the authentication fails, it may be due to an insufficient number of members participating in the reconstruction of the secret or a malicious user attempting to join the group authentication by providing a wrong share. In such cases, the malicious nodes taking part in the authentication should be identified and removed from the scheme. This is explained in the next subsection. Due to the verifiable secret sharing scheme, it is still possible to authenticate other nodes and perform a pseudo group authentication even if the threshold is not reached. This will be discussed in Section 5.2.4.

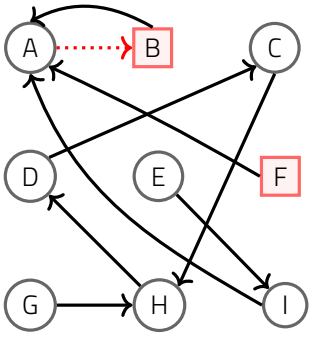
Malicious user identification phase

The malicious node(s) identification is the main contribution of this article. It is an iterative process separating nodes in two categories, either trustworthy or malicious. During the initial step, each node M_i randomly selects another node M_j and assess its authenticity. To do this, the node will assess the authenticity of the share by verifying the following assertion: $H(V, c_j) = V$, where c_j is the share of the group member M_j and V is the verification data computed during the preparation phase. This verification is based on the third assumption of the hash function presented in subsection 5.2.2.



Node	V	X
A	DGE	F
C	DGE	F
D	GE	F
E		F
D	GE	F
G	E	F
H	IE	F
I	E	F
B	CDGEFHI	
F	HIEBCDG	

(a) **Step I :** Left - Directed graph of the verification undertaken by each node . Right - Categorisation of the nodes into trusted or not trusted.



Node	V	X
A	DGEHIC	BF
C	DGEHIA	FB
D	GECHIA	FB
E	IADGHC	FB
G	HDCEIA	FB
H	IEDCAG	FB
I	EADGHC	FB
B	CDGEFHIA	
F	HIEBCDGA	

(b) **Step II :** Left - Directed graph of the verification undertaken by each node. Right - Categorisation of the nodes into trusted or not trusted.

Figure 5.2: Malicious user identification

This operation's result is forwarded to the other nodes in the network, and trust relationships are considered based on the outcome of the verification. If node A trusts node D, and node D trusts node G, then node A can be said to trust node G as well. This is depicted in Figure 5.2 (a) where authentic nodes are represented by black circles and malicious nodes are represented by red squares. Trust relationships are indicated by black continuous arrows, while distrusted relationships are indicated by red dotted arrows. In the example provided, nodes B and F form a coalition, as indicated by the black continuous arrows connecting them.

In the first phase, each node randomly selects another node that attempted the group authentication and verifies its share using the function $H(V, s_i)$. The result is then broadcasted, and nodes are sorted into either the list of trusted (represented by the V column) or not trusted nodes(represented by the X column). For example, if node A directly verifies node D, node D also verifies indirectly node G, and as a result, node G is added to node A's list of trusted nodes.

The process continues until every node is sorted into either of the two categories. If this hasn't been achieved, the first step is executed again, as shown in Figure 5.2 (b). This time, a node is randomly chosen from the list of unsorted nodes, for example, node A might choose between nodes B, H, I, and C. The result of this evaluation is broadcasted and indirect trust/distrust relationships are considered. The process repeats until every node has been sorted for every node in the network. In the example provided, convergence is achieved after only two iterations.

5.2.4 Discussion

In this section, we will discuss the implications, network security properties as well as the implication of the corruption of either the trusted third party as well as the nodes.

The protocol outlined in Section 5.2.3 addresses the issue of authentication in an islanding scenario. The incorporation of Lightweight Verifiable Secret Sharing (LVSS) [36] into the Group Authentication Scheme (GAS) [31] confers several advantages. First, it impedes malicious nodes from hindering a successful group authentication, as they are identified and excluded from the trusted node group during phase three of the protocol.

Second, mutual authentication can occur even if the number of participating nodes falls below the required threshold t, as any node can verify the authenticity of another node's share at any time using the hash function and verification data V. As a result, a one-to-one authentication method can be derived from this group authentication protocol.

Finally, this scheme tackles the Byzantine General problem[6] [37]. In phase three, the presence of malicious nodes does not impede the reconstruction of the secret for the remaining valid nodes. Unlike other consensus-based approaches, this scheme does not require consensus, as each node has sufficient information to verify the authenticity of all other nodes by itself. As depicted in Figure 5.2 (b), malicious nodes do not become part of the trusted group as long as they do not possess a valid share.

Trusted Third Party Compromise

The Trusted Third Party is a pivotal component in the proposed protocol. If it is corrupted during the Preparation Phase, the protocol will be ineffective and the network's integrity during normal operation will also be corrupted. However, in the event of a corruption and disconnection, the microgrid can still operate independently for authentication purposes. Once the Preparation Phase is complete, the Trusted Third Party cannot add a new node to the network unilaterally by generating a new valid share for that node. If it does, it may be able to bypass the group authentication mechanism, but it will not be recognized as valid by the Verificator as presented in Section 5.2.2. The Trusted Third Party would have to update the Verificator with the new share and broadcast the updated information to all nodes. If the nodes do not receive the updated information, the new share would be deemed invalid by the Verificator.

Node Compromise

It was earlier mentioned that a node without a valid share cannot participate in the group authentication. However, if a node with a valid share becomes compromised, both the node and the network could be put at risk. This node may categorize malicious nodes as trusted and broadcast this information to other nodes. Ideally, this would result in inconsistency among various nodes and require a double verification of the mistakenly categorized nodes. In the worst case scenario, the compromised node may be the only one verifying another malicious node, and this information spreads to the other nodes. To mitigate this issue, a double check of each node may be implemented, requiring each node to be verified by at least two other nodes in order to be considered trusted. However, this would increase the computational resources and increase the delay.

5.2.5 Conclusion

In this work, we proposed to apply a group authentication protocol coupled with a lightweight verifiable secret sharing scheme in an islanding scenario in smart grids. This decentralized protocol enhances the robustness of smart grids by enabling smaller sub-grids to continue its operation while maintaining an authentication mechanism. Including the verifiable secret sharing part allowed to strengthen the group authentication by identifying and excluding malicious nodes from the network. Through this protocol, authentication using fewer nodes than the required threshold of a group authentication is possible.

Future research include integrating a Physical Unclonable Function which is a device exploiting a unique randomness to create a digital fingerprint [38]. It would increase the security of the storage of the shares. Additionally, investigating methods for revoking shares to eliminate compromised nodes is another area can be explored. This could be done by either acting on the secret reconstruction or on the verification data.

[6] The Byzantine generals problem consists in finding a way to determine the reliability of the information received from each node in a distributed system in the presence of malicious nodes in order to reach a consensus on a decision.

5.3 System Polymorphism (Vincent Rossetto, Benoît Knott, Kenichi Yasukata)

The concept of polymorphism was initially introduced within the context of networking, referred to as *network polymorphism*. However, as research progressed, it emerged that the core idea behind polymorphism could be extended beyond networks to other systems. This led to the broader concept of *system polymorphism*. The primary objective of system polymorphism is to ensure that different entities, such as users, machines, or processes, have distinct and customized views of the same system. These systems could be networks, file systems, firewalls, or any other cyber-physical infrastructure.

Three types of polymorphic systems have been studied in this research: network polymorphism, cyber-service polymorphism and File system polymorphism.

Among these, file system polymorphism has seen the most significant progress, resulting in tangible outputs. However, the other two, network and cyber-service polymorphism, remain conceptual at this stage, with further research needed to implement these ideas.

System polymorphism aims to control and restrict what entities can perceive or interact with, depending on their operational requirements. It serves as an advanced security paradigm that conceals unnecessary or sensitive information, enhancing resilience against cyber-attacks. By limiting exposure to only relevant entities, it can significantly reduce the attack surface, rendering unauthorized access or malicious activities ineffective. This section explores the applications of system polymorphism across various use cases, starting with *network polymorphism*.

5.3.1 Network Polymorphism

Network polymorphism, a precursor to the broader system polymorphism, focuses on limiting device visibility and communication within a network. In polymorphic networks, devices are only aware of and can communicate with other devices that are essential for their operation. Unnecessary communication paths are not just blocked but made completely undetectable, which increases both the privacy and security of the network. This is especially useful in mitigating potential attacks by reducing the exposure of vulnerable devices to attackers.

Network polymorphism can for example be applied in the context of *Dynamic Dispatching of Cyber-Service-as-a-Service (CSaS)* services. In this scenario, edge agents that handle user requests are responsible for deriving the most appropriate chain of services to process these requests. These agents then dynamically dispatch tasks to available devices within the polymorphic network, as illustrated in Figure 5.3.

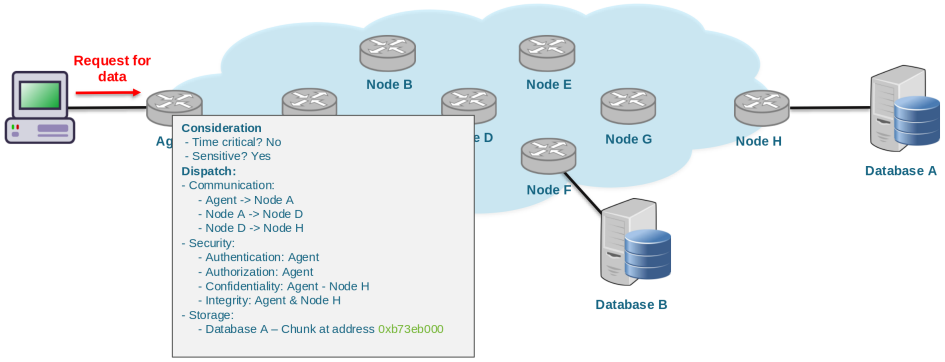


Figure 5.3: Dynamic Dispatching of CSaS Services: Edge agents receiving user requests determine the optimal chain of services for handling the requests and dispatch tasks to available devices.

Techniques for Implementing Network Polymorphism

Several techniques can be considered for implementing network polymorphism, especially within local area networks (LANs). The use of VLANs (Virtual Local Area Networks) offers one possible approach, as VLANs allow for the segmentation of networks into isolated groups. This makes it possible to control which devices can communicate with each other. Specifically, *Private VLANs* [39] and *Asymmetric VLAN mappings* [40] have been studied for this purpose. However, these solutions typically require significant preconfiguration and are often

static, which limits their usefulness in dynamically evolving environments. To overcome these limitations, an over-the-top dynamic configurator could be used to manage VLANs and ensure that only authorized communication paths are established.

Another concept could involve storing a map between devices and their allowable communication partners within a central controller. This device would dynamically manage the network visibility and block unauthorized communication attempts.

Anomaly Detection in Network Polymorphism

Anomaly detection can also play a critical role in network polymorphism by detecting unusual or unauthorized communication patterns. These anomalies could trigger real-time updates to the network's views, ensuring that only legitimate communications are permitted. For instance, a system could automatically update the communication rules of a device if it detects that it is being targeted by an unusual amount of traffic or an unrecognized sender.

5.3.2 Cyber-service Polymorphism

Cyber-service polymorphism is an approach that applies the concept of polymorphism to cyber-services such as firewalls, authentication servers, or other critical components of a cyber-physical system. The key idea is to deploy multiple implementations of a given service at the same point in the infrastructure. In contrast to traditional configurations, which typically rely on a single service or firewall to manage traffic, cyber-service polymorphism introduces variability and redundancy to enhance resilience against cyber-attacks.

This method addresses several challenges associated with the security and robustness of services. By employing multiple distinct instances of a cyber-service, each possibly from a different vendor or with unique configurations, the system can mitigate the risk of a single point of failure. For example, in the context of firewall polymorphism, using multiple firewalls at the same location can help reduce the risk of an attacker successfully exploiting a vulnerability in a single firewall. If one firewall is compromised or overloaded, the others can continue to operate, ensuring that the overall security of the system is maintained.

Advantages of Cyber-service Polymorphism

Cyber-service polymorphism provides several benefits in terms of enhancing system resilience:

- Avoidance of single points of failure: Multiple service implementations reduce the likelihood that a single vulnerability will lead to system-wide compromise. If one service instance is vulnerable, it can be patched while the others continue to provide protection.
- Increased robustness against misconfigurations: Configuration errors are a common source of security vulnerabilities. By deploying multiple services, even if one is misconfigured, the others can act as a backup, limiting the impact of such errors.
- Reduced risk of Denial of Service (DoS) attacks: If one instance of the service becomes overloaded or suffers from a DoS attack, the other instances can continue functioning, ensuring availability.

Challenges and Solutions

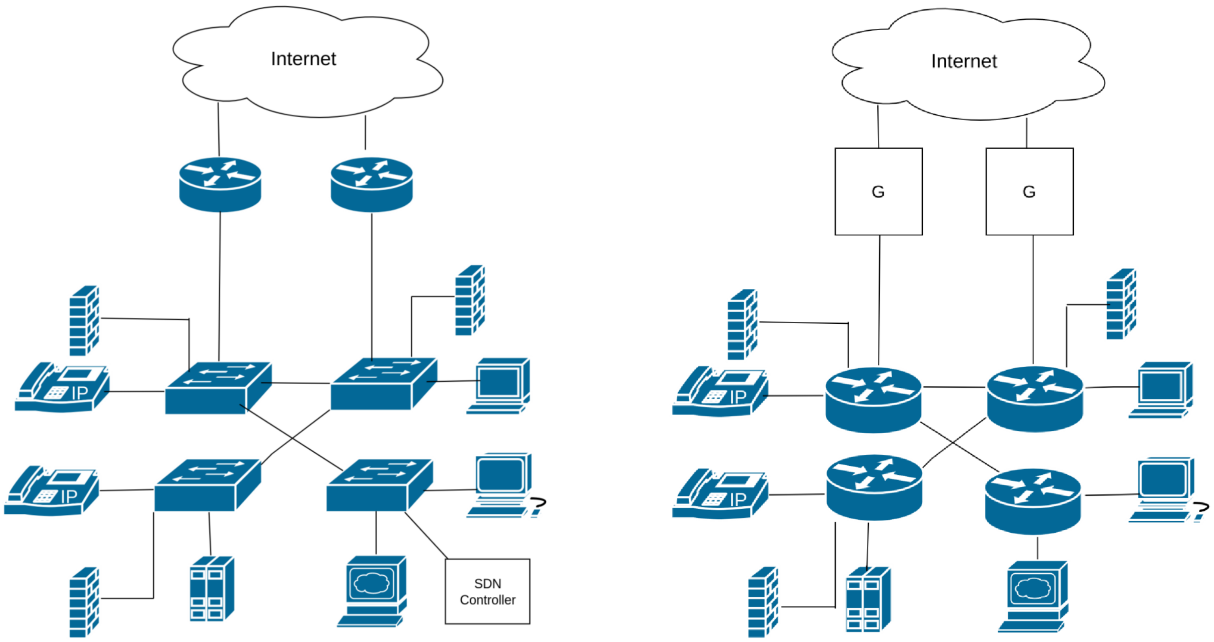
While cyber-service polymorphism provides clear advantages, it also introduces certain challenges that need to be addressed:

- Resource intensity: Running multiple instances of a service can consume more computational and network resources. One potential solution to this is to reduce the amount of duplicated processing.
- Administrative complexity: Managing multiple instances of a service increases the complexity of policies, backups, patching, and general maintenance. Automating configuration management and patch deployment can mitigate this issue.
- Cost: Deploying multiple instances of a cyber-service incurs additional costs in terms of hardware, software licenses, and maintenance. However, these costs may be justified by the increased security and resilience of the system.

Implementation Approaches

The implementation of cyber-service polymorphism involves several strategic decisions related to both the system's architecture and the coordination of service instances. These decisions can be influenced by factors such as whether a load balancer is present to distribute traffic, the physical and logical placement of cyber-services in relation to each other, and how the services are integrated into the network. For instance, packets may be forced to traverse certain cyber-services before entering the network, or the services could be isolated in segments.

Technologies such as Software-Defined Networking (SDN) or segment routing could also facilitate the implementation of cyber-service polymorphism by enabling packet duplication, flexible routing, and arbitration between service outputs (Figure 5.4). For Anomaly detection (Section 5.3.1) information at the gateways or SDN controller could be used to detect faulty replicas.



(a) Software-Defined Networking (SDN)

(b) Segment Routing

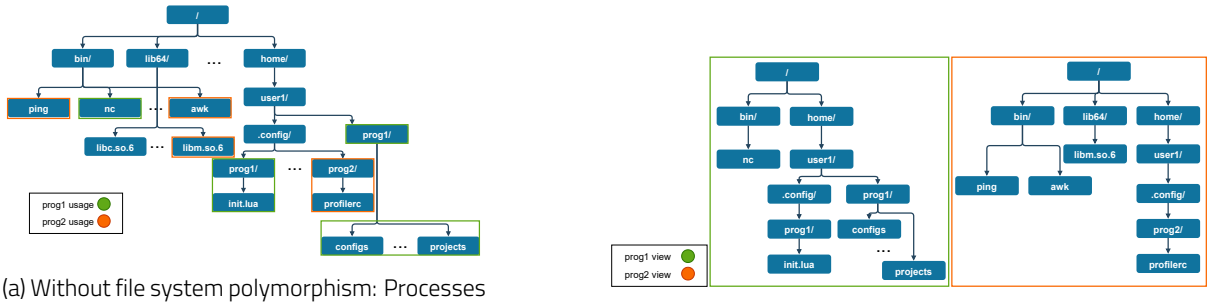
Figure 5.4: Technologies which could provide a mechanism to duplicate packets, route them towards their destination and arbitrate between their outputs. (a) Software-Defined Networking: Logically centralized controller that programs the data plane, i.e. the switches are programmed to forward the packets adequately. (b) Segment Routing: Gateway duplicates the packets and encodes a set of intermediate destinations within the packets themselves.

- Majority-vote algorithm: In scenarios where cyber-services produce binary outcomes (e.g., firewall rules: allow or deny), a majority-vote algorithm can be employed. The system requires a predefined number of instances to reach consensus before allowing a decision to be executed. This approach enhances robustness but may introduce performance overhead due to the coordination required for achieving consensus.
- Asymmetric vs Symmetric approaches: In symmetric configurations, all instances of a cyber-service process the same input identically. In contrast, the asymmetric approach introduces diversity in processing. For example, in firewall polymorphism, one instance might handle the full packet while others process only the header. This selective processing can optimize resource usage, minimizing redundant operations where full processing is not necessary.
- Sequential vs Parallel processing: In a parallel approach, all service instances process input simultaneously. Conversely, in a sequential approach, each service processes the input in turn. Sequential processing can be advantageous when combined with a majority-vote algorithm, as decisions may be finalized early, before all services have completed processing. This can save time and resources. Even in sequential implementations, service instances can operate concurrently but on different inputs or data flows.

5.3.3 File System Polymorphism

File system polymorphism extends the idea of system polymorphism to file systems, aiming to reduce the attack surface by providing processes and applications with isolated, tailored views of the file system. This concept is particularly important in environments where programs often have access to files they do not require, which increases the risk of exploitation in case of a security breach. The goal of file system polymorphism is to prevent unnecessary file exposure by restricting file access based on the application or process interacting with the system, rather than solely on user permissions.

Conventional file permission models, such as POSIX and ACL, primarily control access based on user roles. However, these models are less effective at safeguarding against processes running under the same user context. This is particularly problematic in personal computers or multi-user environments like servers, where one compromised application can lead to unwanted data exposure or system vulnerability. File system polymorphism directly addresses this concern by providing processes with their own customized views of the file system.



(a) Without file system polymorphism: Processes have access to the entire file system, leading to potential security risks

(b) With file system polymorphism: Processes are provided with isolated views of the file system

Figure 5.5: Comparison of process access rights, or views of the file system, in different environments

In Figure 5.5, we illustrate file system polymorphism by comparing the access rights, or the views of the file system, of two processes in a standard environment versus an environment with file system polymorphism implemented.

Approaches to File System Polymorphism

Different strategies have been explored to implement file system polymorphism. These approaches balance security, usability, and system performance, each providing varying degrees of isolation and flexibility.

Partial Segregation Approach In this approach, different applications or processes are provided with segregated views of the file system, though some views can overlap. For maximum security, applications could interact with completely separate file systems, whether the separation is physical (i.e., physically isolated on the disk) or logical (managed by software). This isolation reduces the risk of privilege escalation and unintended data sharing between processes.

However, implementing this strict segregation requires kernel or file system-level modifications. To improve portability and ensure the approach can be deployed across diverse systems, an "over-the-top" method has been proposed. This would leverage existing user and group mechanisms present in most operating systems.

Managing file access in this model can be achieved through two strategies (or a combination of both):

1. User-Based Permission Prompts: When an application attempts to access a file outside its permitted view, the system could prompt the user to grant access, similar to permission requests on mobile platforms like Android or iOS.
2. AI/ML-Driven Access Control: Artificial intelligence or machine learning algorithms could dynamically determine which processes are allowed to access which files, automatically adjusting permissions based on usage patterns. This could include broad access rules, such as allowing a program access to all files of a certain type (e.g., `.jpeg` images) or to all files within a specific directory (similar to the sandboxing approach used by Flatpak).

This method aims to balance usability and security while allowing for flexible file access management.

Full Segregation Approach This approach goes a step further by ensuring that any file appearing in multiple views is treated as a separate instance for each process. Changes made in one view do not affect others, and modifications must be manually merged if needed. While this guarantees maximum isolation, it introduces complexity in managing file versions and handling merges, which may not be intuitive for users.

Versioning Approach In this less restrictive approach, file access is not limited, but any modification made by a program is versioned. This allows users to revert to previous versions of a file if unwanted changes are introduced (such as a ransomware attack). Although this approach doesn't prevent unauthorized access, it enhances security by preserving file integrity over time.

PolymorFS

PolymorFS represents a significant advancement in file system polymorphism through its versioning approach, which addresses the limitations of traditional storage systems, especially in terms of storage capacity and version management. Traditional versioning systems, such as Copy-on-Write (CoW) storage, often face challenges due to the limited physical storage available. They rely on the deletion of older versions to free up space, which poses a risk for critical data that might be needed after the fact. PolymorFS tackles this by leveraging the virtually limitless storage capacity of the cloud, allowing for the retention of extensive version histories without concern for space constraints.

PolymorFS introduces online file versioning, recording every file change at the granularity of 4 KB. This ensures that every write operation is tracked, providing fine-grained control over file modifications. By maintaining consistent and crash-safe logs, PolymorFS guarantees that no version history is lost, even in the event of an unclean system shutdown. Its architecture is designed to be independent of the underlying file system, making it deployable across both on-premise systems and cloud-based virtual machines. This modular design eliminates the need to replace the operating system kernel or file system, enhancing its deployability and ease of use.

One of the primary benefits of PolymorFS is its ability to store version histories as regular files, which can be easily transferred to external cloud storage. This portable format not only facilitates the seamless transfer of version data but also allows for their simple deletion once they are safely stored in the cloud. The system synchronizes the storage of original file data with version logs, ensuring metadata consistency and preventing data loss during the synchronization process.

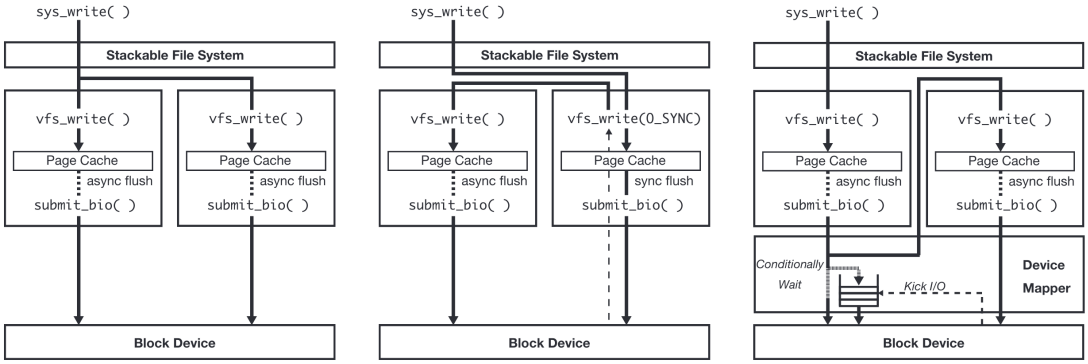
PolymorFS is particularly useful for mission-critical applications, such as databases, where frequent updates occur, and maintaining a history of every write is essential. Unlike traditional CoW systems, PolymorFS reduces the performance overhead typically associated with versioning by implementing a parallel I/O optimization strategy (see Figure 5.6). This technique allows PolymorFS to write version logs and original file data simultaneously, minimizing latency and ensuring that performance is not significantly impacted.

In summary, PolymorFS offers a highly scalable, efficient, and cloud-compatible versioning storage solution that can meet the demands of modern cyber-physical systems. By combining fine-grained versioning with cloud storage, it provides an innovative tool to the challenges of file system polymorphism based on the versioning approach.

LogCache

LogCache, another system developed during this research, focuses on optimizing write performance in flash caches at the file system layer. While not directly a polymorphic file system, LogCache complements file system polymorphism by addressing the performance bottlenecks often encountered in write-heavy environments, especially in scenarios where fast, consistent writes are necessary, such as in virtualized environments or databases. Typical block layer caches, while providing portability and transparency, often suffer from increased I/O overhead due to redundant crash-safe metadata manipulation occurring at both the file system and block layer levels. This redundant metadata management significantly impacts latency, especially in write-intensive workloads (see Figure 5.7 for an architectural overview).

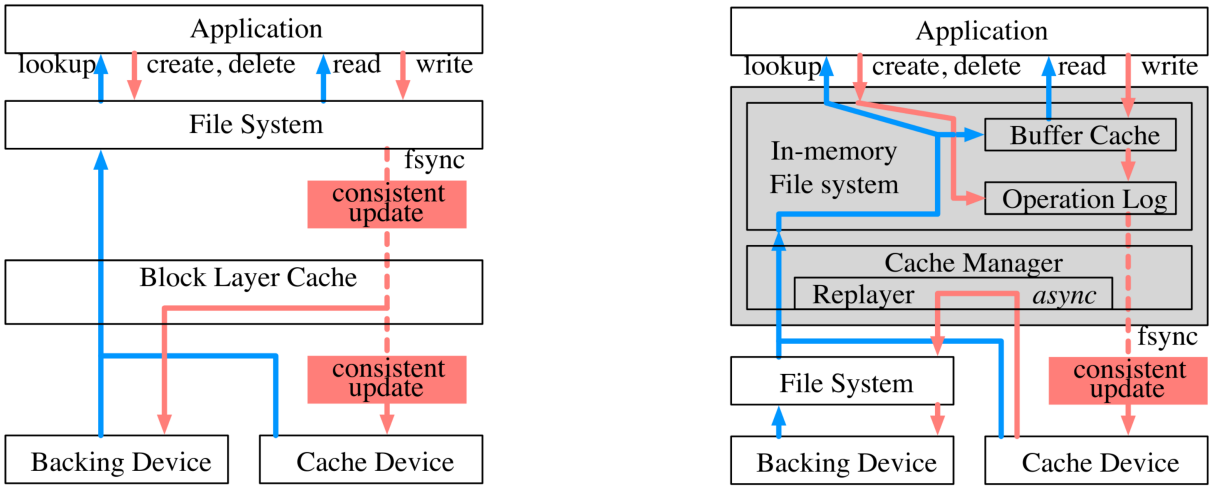
LogCache improves upon this by decoupling the file system metadata operations from the latency-critical write path, allowing for more efficient handling of storage I/O. The cache operates at the file system layer, maintaining compatibility with existing kernel file systems without requiring source code modifications. This portability ensures that *LogCache* can be incrementally deployed in a wide range of systems.



(a) Simple file system layer versioning (b) O_SYNC (c) PolymorFS

Figure 5.6: Stackable file system based versioning. The first case (Figure 3a) does not synchronize the progress of file system writes between original and log data. Subsequently, unclean shutdown may cause version lost. The second case (Figure 3b) performs file system write for the log file with the `O_SYNC` option so that it can assure that the written data become persistent after the return from the file system write. And also every file system write for original data is blocked in order to wait in order to synchronize the progress of data persistency. PolymorFS (Figure 3c) can normally perform block writes for the original without waiting for the log data. Persistency is controlled at the device-mapper and the synchronization mechanism will block the write for the original data if necessary.

The system achieves its performance enhancements through a stackable file system interface, which allows file system operations to be redesigned specifically for cache devices. By separating file system metadata from the cache write path, *LogCache* reduces the write latency that typically arises from concurrent metadata updates in block layer caches. Additionally, *LogCache* introduces a highly optimized `fsync` design that minimizes the overhead typically associated with small, random writes, as shown in Figure 5.8. This is particularly beneficial in multi-threaded environments, as it ensures higher scalability and better throughput under concurrent workloads (see Figure 5.9).



(a) Block layer cache

(b) LogCache

Figure 5.7: LogCache architecture. Red allows are part of the write path, and blue arrows represent the read path. Dashed arrows are part of fsync. LogCache is shown as the shaded box in Figure 5.7b

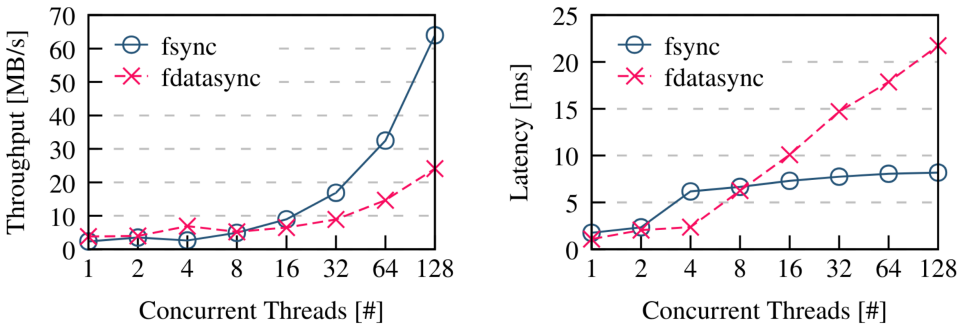
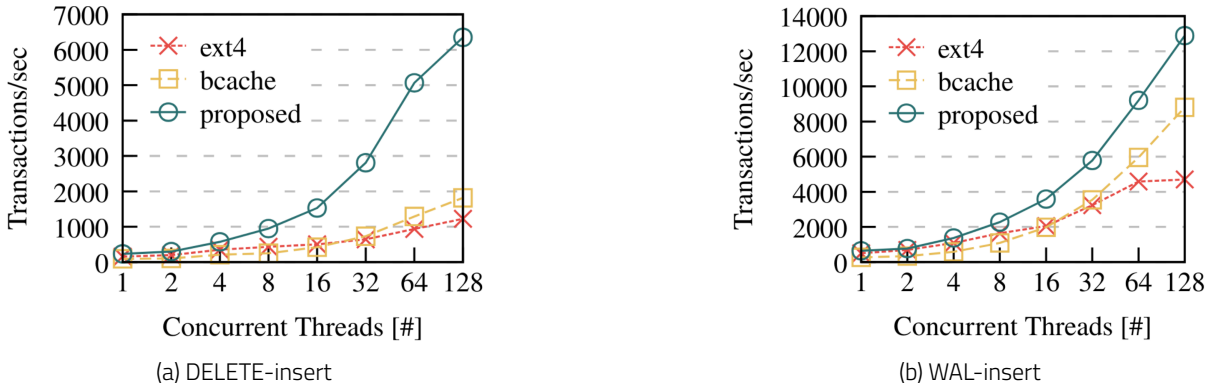
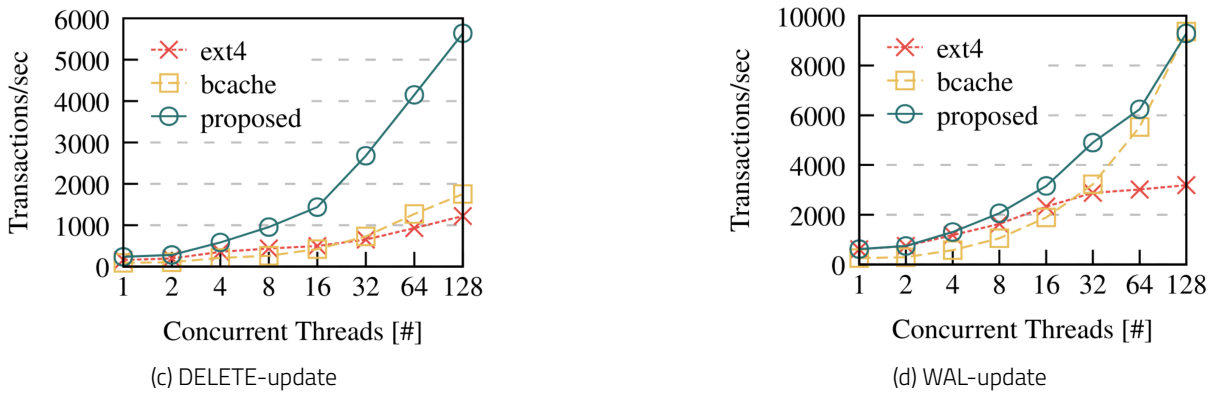


Figure 5.8: ext4 fsync and fdatasync performance on an NVMe SSD with varied concurrency.



(a) DELETE-insert

(b) WAL-insert



(c) DELETE-update

(d) WAL-update

Figure 5.9: Mobibench SQLite benchmark.

For polymorphic file systems, *LogCache* provides a specialized mechanism for write-optimized flash caching, contributing to the overall objective of isolating and customizing file system views while maintaining performance efficiency. By doing so, *LogCache* can help enhance the resilience of file systems by offering efficient, crash-safe storage without the latency costs typical of traditional methods.

Experimental results, including both micro-benchmarks and real-world application tests, show that *LogCache* delivers substantial performance gains in throughput and latency compared to traditional block layer caches. In particular, *LogCache* avoids the redundant metadata management found in caches like bcache, which introduces significant write latency due to double metadata handling at both the block and file system layers. By eliminating this inefficiency, *LogCache* achieves approximately three times the performance of conventional block layer caches while maintaining transparency and broad applicability.

Figure 5.10 compares latency between an NVMe SSD-only configuration and bcache, emphasizing the increased latency that bcache incurs (see Table 5.1 for a detailed breakdown of latency factors). This comparison highlights the performance bottlenecks that *LogCache* is designed to overcome, particularly under heavy write conditions, making it well-suited for use cases like databases that require low-latency and crash-safe operations.

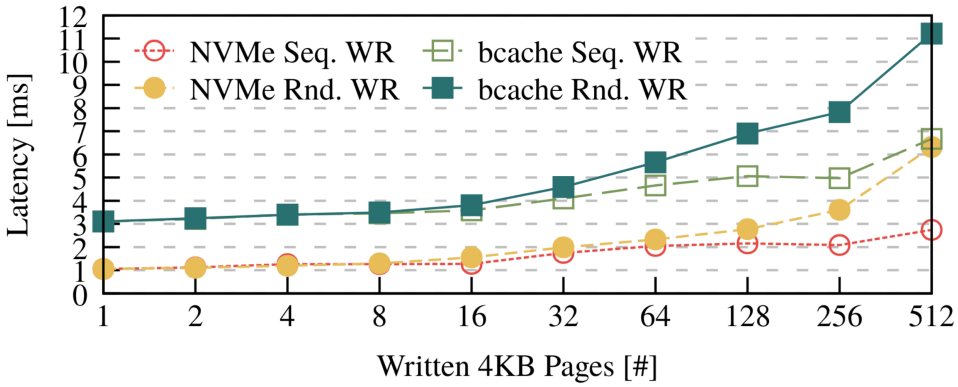


Figure 5.10: Write latency on an NVMe SSD and bcache.

	NVMe SSD			bcache		
Component in Write	1 page	Seq. 256 pages	Rnd. 256 pages	1 page	Seq. 256 pages	Rnd. 256 pages
Send dirty pages	3.332	240.552	1066.725	9.868	876.495	1862.528
Wait on pages	14.608	233.449	66.849	29.454	376.217	927.860
Flush command	1040.018	1522.109	1730.492	3068.365	3620.208	4330.333
Total	1057.958	1996.110	2864.066	3107.687	4872.920	7120.721
Per-page cost	1057.958	7.797	11.187	3107.687	19.034	27.815

Table 5.1: Write latency breakdown on an NVMe SSD and bcache. Results are reported in microsecond scale.

Although LogCache's primary goal is performance optimization, its deployment at the file system layer and its ability to seamlessly integrate with existing kernel file systems make it a valuable addition to environments where file system polymorphism is implemented. The increased write efficiency it provides helps maintain system performance even when managing complex file operations and versioning mechanisms, as seen in systems like PolymorFS.

5.3.4 Future Work

Network Polymorphism

A significant avenue for future work is to explore the use of VLANs in implementing dynamic polymorphic networks. Specifically, experiments could be conducted to determine how easily VLANs can be reconfigured dynamically and whether such approaches can be extended to support polymorphism at scale. Investigating the feasibility and limitations of VLAN-based polymorphism, particularly in cases where dynamic network segmentation is required, will be an essential next step in this research.

Cyber-service Polymorphism

Future research in cyber-service polymorphism should focus on enhancing the performance and scalability of these systems. For example, techniques could be developed to reduce the resource footprint by intelligently selecting which services process certain types of traffic, or by dynamically adjusting the number of active service instances based on system load and threat level.

Additionally, research is needed to explore interoperability between different service implementations. For instance, if one instance of a cyber-service needs to be restarted or replaced, it would be beneficial for the system to seamlessly transfer its state to another instance, even if they are from different vendors. Achieving such state translation or synchronization across heterogeneous systems will be a key challenge in making cyber-service polymorphism more practical for widespread deployment.

File System Polymorphism

Despite promising developments, a fully realized, universally deployable polymorphic file system has yet to be implemented. While solutions such as PolymorFS and LogCache offer modular approaches to file versioning and write optimization, they stop short of providing comprehensive polymorphism. Instead, these tools serve as a foundation, showcasing specific mechanisms that could support file system polymorphism but not fully achieving it. The path to a fully polymorphic file system requires additional research and development, particularly in balancing security, usability, and system performance within existing operating system constraints.

One of the main challenges in implementing file system polymorphism is ensuring that it consistently applies across all actions involving file access. Triggering polymorphism requires the system to respond intelligently when applications attempt to interact with files they do not have explicit access to, whether through prompting users for permission, copying files to isolated views, or creating new file versions. To trigger polymorphism appropriately during file interactions, some possibilities include leveraging the use of the Virtual File System (VFS) [41] in Linux or using a File System MiniFilter [42] in Windows. These mechanisms can help intercept file operations and apply necessary transformations, allowing for a flexible, context-aware approach to file system behavior.

Additional complexities arise in environments where applications are installed, updated, or uninstalled, as each of these actions could affect the structure of the polymorphic file system and require dynamic adjustments to file permissions and access rights.

Future research may focus on enhancing the automation of file access control, using machine learning to dynamically assign file permissions or views based on application behavior, or improving the performance of polymorphic file systems in environments with constrained resources, such as IoT devices.

5.4 Supply Chain Insecurity: The Lack of Integrity Protection in SBOM Solutions (Can Özkan, Xinhai Zou, Dave Singelée)

5.4.1 Introduction

Modern software systems involve increasingly complex and dynamic supply chains. Lack of visibility into the composition and functionality of these systems contributes to cyber security risks [43]. The heavy reliance on third-party software components and dependencies within the modern software ecosystem necessitates robust strategies to ensure software supply chain security. One notable example is the SolarWinds attack, which exploited weaknesses in the software update mechanism and highlights the critical need for organizations to have better visibility into their software dependencies [44]. SBOMs have emerged as a crucial instrument for improving software transparency and visibility. An SBOM is a formal, machine-readable inventory of software components and dependencies, information about those components, and their relationships [45]. It provides an inventory of the various software components used in an application, such as libraries, frameworks, and dependencies. This nested inventory identifies the root and nested components within the software, allowing for better management and mitigation of potential security risks. [46]. By providing a comprehensive inventory of all open-source and proprietary components, along with their respective licenses and vulnerability information, SBOMs empower organizations to manage security, license, and compliance risks effectively in the ever-evolving software development ecosystem [47].

The adoption of SBOMs becomes vital after the Biden administration signed the Executive Order on Improving

the Nation's Cybersecurity in May 2021 [48], [49]. This order emphasizes the need for secure software development practices and forms the basis for standardizing and promoting the use of SBOMs across government agencies and critical infrastructure sectors. The executive order mentions that commercial software development often lacks transparency, sufficient focus on the software's ability to resist attacks, and adequate controls to prevent tampering by malicious actors. Implementing more rigorous mechanisms is urgently needed to ensure products function securely and as intended. For this reason, any company that sells software to the federal government must provide the application and a corresponding SBOM [50]. Since SBOM is mentioned in the Cyber Resilience Act [51] published by the European Parliament in July 2023, it is expected to become mandatory in the EU in the upcoming years as well.

There is a relationship between entities producing and acquiring software. We denote the entity that produces software as supplier in the rest of this work. On the other hand, we denote the entity that acquires and purchases the software from the supplier as consumer. The responsibility for generating an SBOM typically lies with the supplier [43]. This includes, but is not limited to, compiling an inventory of all components, libraries, and dependencies used in the software, along with their corresponding versions and license information.

An SBOM encompasses various formats for documenting and sharing information about software components within the supply chain. Two prominent SBOM types, CycloneDX and SPDX (Software Package Data Exchange), have emerged as leading standards in this domain. SPDX, created by the Linux Foundation, is an open standard for communicating SBOM information, including provenance, license, security, and other related information [52]. Its main use case is license compliance. CycloneDX, on the other hand, was primarily designed by OWASP to solve vulnerability identification, license compliance, and outdated component analysis [53].

Let us now zoom in on vulnerability management. An SBOM facilitates a systematic approach for identifying security weaknesses by providing a detailed inventory of software components and their versions. This process can be implemented proactively at regular intervals, with a frequency depending on the organization's policy, or reactively in response to specific vulnerabilities. Vulnerability management can be performed regularly by checking the SBOM against a vulnerability database such as the National Vulnerability Database (NVD) to identify any known vulnerabilities in the software components. Therefore, potential security risks can be proactively identified and addressed. This process can be automated to ensure timely and efficient vulnerability management. In addition, vulnerability management utilizing SBOM can be triggered at specific times, such as when a new vulnerability with a sufficiently high level of CVSS (Common Vulnerability Scoring System) is discovered. In such cases, the SBOM can be quickly referenced to determine if the affected software component is used in the organization's software system, allowing for swift vulnerability identification, remediation, and risk mitigation. For instance, when Log4j, also known as Log4Shell with CVE-2021-44228, was first discovered, it created widespread concern due to its potential to be exploited by adversaries who request arbitrary LDAP and JNDI servers, resulting in the execution of arbitrary code remotely or sensitive information leakage [54]. The vulnerable Java components can be quickly searched in an SBOM, and the organization can quickly determine if they are affected. This incident emphasized the urgent need for organizations to assess their exposure to a critical vulnerability and take appropriate remediation actions immediately.

Both in vulnerability and license management, the integrity of SBOMs is of critical importance. In this work, we exhaustively investigated the consumption tools and generation processes of SBOMs for prominent software languages. This involves examining how adversaries could potentially manipulate SBOMs to hide security vulnerabilities. We explored how tampered SBOMs might deceive consumption tools into producing inaccurate vulnerability data, resulting in organizations overlooking vulnerabilities. We also explored how generation tools could be attacked to provide incorrect and misleading dependency information leading to the concealment of vulnerabilities when consumed. In summary, such manipulations can lead to the generation of misleading vulnerability reports by SBOM consumption tools, resulting in organizations to inadvertently believe that their software assets are free from vulnerabilities. Therefore, organizations may overlook critical security risks, exposing themselves to potential exploitation and compromise. Our investigation underscores the crucial importance of robust integrity controls and verification mechanisms within the SBOM life cycle to ensure the accuracy and reliability of SBOM data.

The rest of Section 5.4 is organized as follows: In Section 5.4.2, we describe the life cycle of an SBOM, explain the software development lifecycle (SDLC), and define possible attack points. In Section 5.4.3 and 5.4.4, we investigate SBOM consumption and generation tools, respectively. Section 5.4.5 presents a solution to mitigate the identified integrity problems and discuss the details in Section 5.4.6. We discuss related work in Section 5.4.7 and conclude our work in Section 5.4.8.

5.4.2 Lifecycle of an SBOM

Due to the strong relationship between SBOM and the accompanying software, we opt to first briefly discuss the Software Development Life Cycle (SDLC) before moving to the SBOM lifecycle.

Software Development Life Cycle (SDLC)

The Software Development Life Cycle (SDLC) is a fundamental component of the software engineering process, detailing the transformation of code into the final product. For a typical SDLC, it consists of six primary phases: requirement, design, implementation, testing, deployment, and maintenance [55]:

1. ***Requirement:*** *This phase involves gathering requirements from stakeholders to assess the product's feasibility, revenue potential, and cost.*
2. ***Design:*** *This phase includes system architecture, selection of programming languages, and security measures to mitigate potential issues.*
3. ***Implementation:*** *This phase translates the software requirements into code.*
4. ***Testing:*** *This phase validates the software's functionalities, user experience, and security.*
5. ***Deployment:*** *This phase releases the final product and delivers it to end-users.*
6. ***Maintenance:*** *This phase monitors the entire software life cycle to ensure any issues are identified and corrected in subsequent updates.*

Due to the extensive benefits of using the SDLC, such as structured approach, risk management, and cost-effectiveness, many organizations are adopting SDLC in order to improve their software development process [56].

SBOM-integrated SDLC

To enhance the security and visibility of the supply chain, the National Telecommunications and Information Administration (NTIA) introduced the concept of the Software Bill of Materials (SBOM) in 2018 [57]. Due to the strong connection between SBOM and the SDLC, discussing the SBOM lifecycle in conjunction with the SDLC is essential. Fig. 5.11 illustrates an SDLC integrated with SBOM, featuring a typical SDLC on the supplier's side, the distribution stage, and a maintenance phase on the consumer's side. This section will focus on the SBOM lifecycle, excluding the typical SDLC components detailed in Section 5.4.2.

Throughout the development life cycle, the SBOM helps track and manage dependencies and ensures that all components meet security and licensing requirements on the supplier's end. Suppliers can use the SBOM to track and manage software dependencies in the implementation phase. This ensures that all components are up-to-date, secure from known vulnerabilities, and compliant with licensing terms. The supplier can continuously update the SBOM to reflect changes in dependencies or libraries, which ensures that the final delivered SBOM accurately reflects the software's composition. The software undergoes various tests in the testing phase, such as penetration testing, code review, source composition analysis, and dynamic security testing. The SBOM facilitates identifying potential vulnerabilities in dependencies on the supplier's end. Once the testing phase is fulfilled, it is deployed to the production environment. When deploying the final product, the supplier can include the final SBOM as part of the deliverables. This provides the consumer with detailed information about the software's composition and dependencies. After deployment, the software enters the maintenance phase. The consumer can then use the delivered SBOM to perform periodic vulnerability checks. By comparing the components listed in the SBOM against vulnerability databases like the NVD, the consumer can identify known vulnerabilities in either a proactive or a reactive way.

Trust assumption in SBOM lifecycle

Before we zoom in on the security of the SBOM lifecycle, we first need to specify our trust assumptions for suppliers, consumers and the communications channels they rely on. We want to minimize these trust assumptions. Therefore, we do not make any assumptions regarding the security of the file systems, used to store SBOMS and/or software, used by suppliers and consumers. Moreover, no trust assumptions are made on the security of the communication channel between suppliers and consumers. The only trust assumption we make, is that any key management system used by either consumer and/or supplier is reliable and secure.

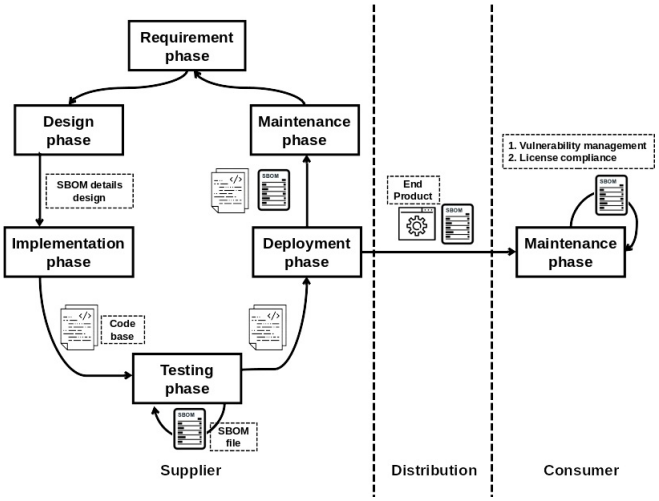


Figure 5.11: Typical SDLC integrated with SBOM

We consider attackers with the goal to compromise the integrity of an SBOM. Based on the trust assumptions above, the attacker has the following capabilities:

- The attacker can modify any file (e.g., SBOM) between supplier and consumer.
- The attacker can alter the codebase at the supplier's side.
- The attacker cannot alter the codebase at the consumer's side[7]
- The attacker has no access to any cryptographic keys either stored by consumer or supplier.

Attack points in SBOM lifecycle

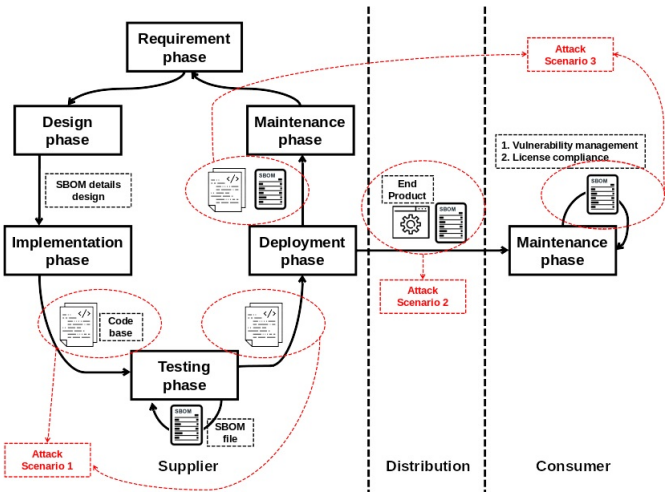


Figure 5.12: Attack points in a typical SDLC integrated with SBOM

Based on the trust assumptions outlined for the SBOM lifecycle, Fig. 5.12 depicts an SBOM-integrated SDLC, highlighting potential attacks at different phases. The figure shows three essential scenarios where attacks might occur: during SBOM generation before the deployment phase (Attack scenario 1), throughout the SBOM distribution process (Attack scenario 2), and during SBOM storage in the maintenance phase (Attack scenario 3).

Attack scenario 1 during SBOM generation before the deployment phase In this scenario, attackers with access to the codebase aim to mislead the SBOM generation process to produce an inaccurate SBOM, affecting

[7]The main reason is that if an attacker is capable of altering the codebase at the consumer's side, it does not really make sense to manipulate the SBOM itself. Therefore, this attack is out of scope of this work.

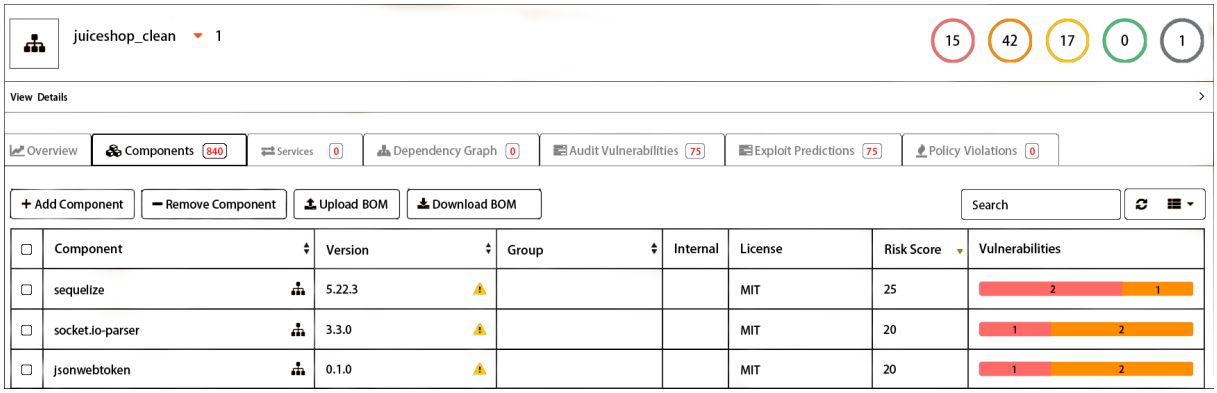


Figure 5.13: Benign SBOM Consumption: Non-tampered SBOM

the results of SBOM consumption on both the supplier's and consumer's sides. To generate an incorrect SBOM, attackers can modify either (1) the source code itself or (2) the library information file, depending on their level of access to the codebase.

1. Source Code Access: If attackers have access to the source code, they can directly inject malicious code into the source code without altering the library information file. In this case, a naive SBOM generator that relies only on the library information file will generate an SBOM reflecting an incorrect product structure unrelated to the source code.
2. Library Information Access: If attackers only have access to the library information, they can modify the library information file to hide vulnerable versions and make these appear as vulnerability-free versions without changing the source code. Similarly, a naive SBOM generator that relies solely on the library information will thus help attackers conceal vulnerabilities.

Attack scenario 2 throughout the SBOM distribution process In this scenario, attackers with access to insecure channels between suppliers and consumers aim to spoof consumers with incorrect information of the end product. By tampering with the library information in the SBOM, attackers can create incorrect information during the consumer's SBOM consumption process for malicious purposes, such as hiding vulnerabilities.

Attack scenario 3 during SBOM storage in the maintenance phase In this scenario, attackers with access to the file system aim to mislead the SBOM consumption process to obtain wrong results. This can happen on both the supplier's and consumer's side. Based on the types of files and the entity being attacked, three different attack scenarios can be identified:

1. Supplier's Codebase Access: If attackers have access to the codebase during maintenance, they can tamper with the source code or alter the library information file based on their accessibility to the code: Source Code Access or Library Information Access. However, these modifications will not affect the current product but compromise future iterations.
2. Supplier's SBOM File Access: If attacks can access the SBOM file on the supplier's side, they can update the version information in the SBOM file when new vulnerabilities are discovered before developers notice. This concealment prevents developers from updating the software to address the vulnerabilities.
3. Consumer's SBOM File Access: If attackers obtain access to the SBOM file on the consumer's side, they can employ the same strategy as on the supplier's side to hide new vulnerabilities. Consequently, consumers will not take any actions to mitigate any hidden vulnerabilities.

5.4.3 Consumption of an SBOM

In this section we discuss our experiments to compromise the integrity of SBOM consumption and, more specifically, tampering with version numbers of dependencies listed in the SBOM. This can have significant security implications. For instance, tampering with values in dependencies can misrepresent the actual dependencies being used, which, in turn, affects the vulnerability management process because it can result in false security assessment results. Therefore, organizations can overlook vulnerabilities and assume they are safe from par-

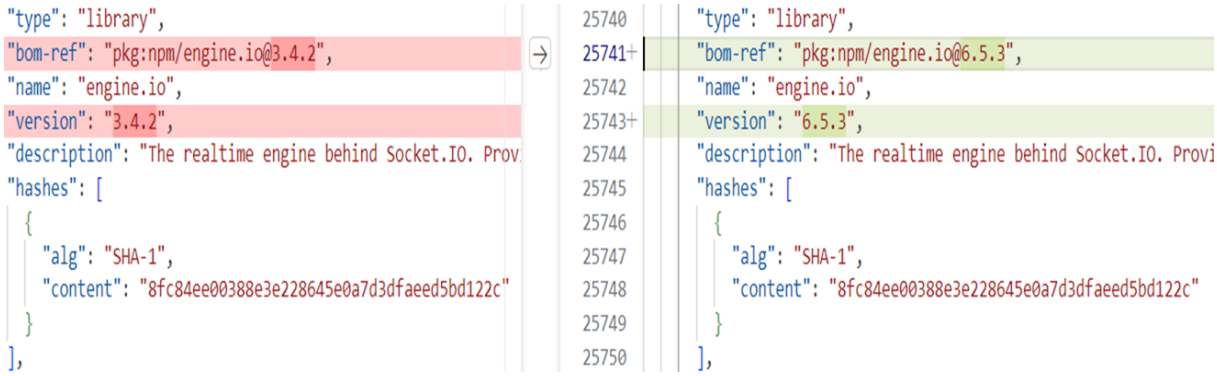


Figure 5.14: Tampering SBOM File without Altering Hash Values

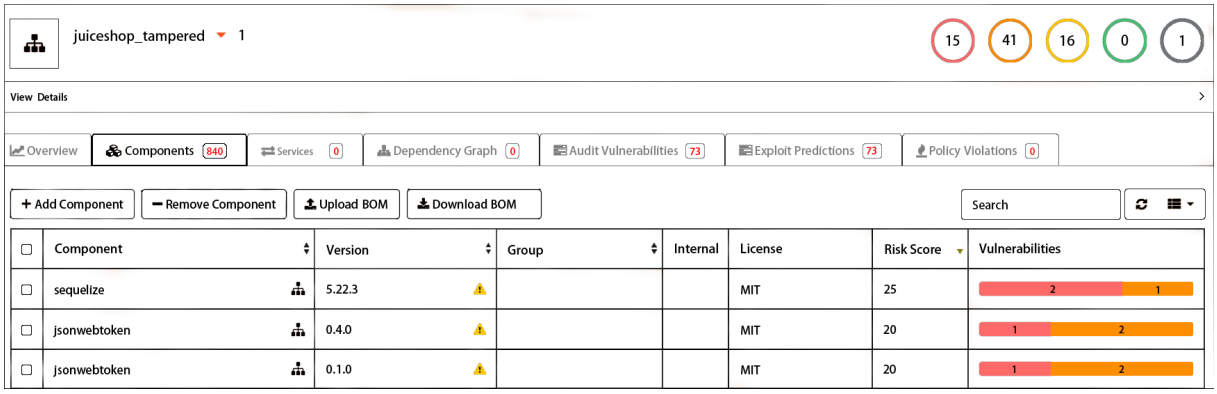


Figure 5.15: Malicious SBOM Consumption: Tampered SBOM

ticular vulnerabilities while they may not be secure. In addition, tampering with license information in an SBOM file can violate your license compliance process, leading to non-compliance with licensing obligations.

We systematically reviewed four distinct consumption applications and analyzed how these tools react when the version numbers are tampered with. As our objective is security, we chose the tools recommended in the CycloneDx tool center. The tools we experimented with are OWASP Dependency Track [58], Bomber [59], cve-bin-tool [60], and Grype [61]. The criteria governing tool selections stem from their adherence to fundamental principles such as open-source availability, unrestricted free accessibility, and formal recognition by the CycloneDX standard. Our analysis revealed a vulnerability that compromises the integrity of dependency information within all four applications. None of the applications employed any cryptographic controls for integrity verification in the dependencies listed in the SBOM file, underscoring a significant gap in integrity control practices. While hash values were present within the SBOM file, they were not checked to ensure the integrity of the libraries and dependencies or to detect potential manipulation. Therefore, our findings show a limitation in existing SBOM consumption tools. These tools rely solely on version number information to identify possible vulnerabilities concerning dependencies, which is a critical shortcoming.

For illustration purposes, we implemented our attack for the OWASP Dependency Track. As discussed above, our code review showed that the tool exclusively focuses on comparing the version numbers of each software component against a vulnerability database. The lack of hash value comparison in this tool was surprising to us. We implemented our attack as follows: We first installed the tool. Then, we created an empty project and filled in the details, such as the project name and type. We used the Owasp-developed Juicy Shop application for demonstration purposes. Upon completion, we imported a clean and non-tampered SBOM file in CycloneDx format. We analyzed the results executed by the tool, which summarizes the benign use case of SBOM consumption. As shown in Fig. 5.13, the vulnerability results are 15 critical, 42 high, 17 medium, and 0 low, with 75 exploit predictions and audit vulnerabilities. Additionally, the tool shows that there are 840 components in the application.

The engine.io library used in the application was manipulated from version 3.4.2, which contains vulnerabilities, to version 6.5.3 without any vulnerability at the time of writing, and we did not change any hash value, leaving the tool a hint to catch the modification. In other words, the SBOM file was tampered with in such a way

that the version number of the engine.io dependency was modified from 3.4.2 to 6.5.3 while leaving the hash value intact, as seen in Fig. 5.14. Subsequently, the newly modified SBOM file was imported into the OWASP Dependency Track. After import, the tool successfully digested the file without checking the integrity of the component we manipulated. Then, it analyzed the file and produced results showing fewer vulnerabilities than the benign use case. More specifically, the tool showed one less high and medium vulnerability this time, as seen in Fig. 5.15, resulting in 41 high and 16 medium, with 73 exploit predictions and audit vulnerabilities. Despite the availability of hash values, we can conclude that the SBOM consumption process lacks integrity control for the dependencies listed in the SBOM file.

Additionally, an unexpected consequence emerged while analyzing tools for SBOM consumption. Each tool relied on a distinct vulnerability database, leading to discrepancies in the number of vulnerabilities identified following SBOM consumption. In particular, Dependency Track can locate 75 vulnerabilities, with 15 critical, 42 high, 17 medium, and one unassigned. In contrast, Bomber can locate 97 vulnerabilities in total, with 16 critical, 39 high, 40 moderate, and two low, and cve-bin-tool can detect 85 CVEs in total, with 22 critical, 63 high, 41 medium, zero low, and six unknown. OWASP Dependency Tracker utilizes the NVD and the GitHub Advisory database as primary sources [62]. Conversely, Bomber leverages the Open Source Vulnerabilities (OSV) database [63] by default. The cve-bin tool, on the other hand, prioritizes vulnerability information from NVD, OSV, Gitlab Advisory Database (GAT), RedHat Security Database (REDHAT), and Curl Database (Curl) sources [64].

Another shortcoming we identified, was that the tools in question do not require SBOMs to be digitally signed before consumption, which presents a security risk. Consequently, even when suppliers digitally sign an SBOM file to confirm its integrity and authenticity, adversaries could effortlessly tamper with it and remove the signature before consumption. This manipulation could lead to excluding specific dependencies or misleading version numbers, potentially obscuring existing vulnerabilities. To address this risk, future iterations of SBOM consumption tools should introduce the enforcement of mandatory digital signatures before consumption.

5.4.4 Generation of an SBOM

The SBOM generation process entails creating a comprehensive list of all software components, such as libraries, frameworks, and dependencies, and their corresponding versions used in an application. This typically involves scanning the software's source code and file system, where these components and other relevant artifacts reside. Available automated tools can scan the source code and file system in order to reveal dependencies, libraries, frameworks, and other components used in the application. In addition, many modern programming languages and frameworks utilize package and dependency management tools such as Maven [65], npm [66], pip [67], and composer [68] that maintain a list of external dependencies imported and their versions so that those dependencies can be easily installed at the beginning of the software installation or when another developer forks the code base. SBOM generation tools, programs that help organizations produce their SBOM, can deduce information to identify and document dependencies through the aforementioned methods.

This section discusses a comprehensive analysis and experimental evaluation of SBOM generation processes across various prominent programming languages. We seek to answer whether adversaries can manipulate the generation process, which could yield inaccurate SBOM files. Manipulating the SBOM generation process to exclude specific dependencies or to mislead version numbers compromises the integrity and accuracy of the resulting SBOM artifact.

Therefore, we conducted experiments to compromise the integrity of the SBOM generation process by manipulating dependency version information in package managers. More specifically, we analyzed SBOM generation for Python, C, C++, Java, C#, Javascript, and PHP—the first seven programming languages based on Tiobe's November 2023 ranking [69]. As the objective of our study is geared toward security, we chose the tools from the CycloneDx tool center. The tools we experimented with for this section are CycloneDx for Python [70], CycloneDx for Conan [71], CycloneDx for Maven [72], CycloneDx for .NET [73], CycloneDx for npm [74], CycloneDx for PHP Composer [75], and Syft [76].

We performed a source code review to reveal the SBOM generation process and how the tools identify dependency information. We carefully reviewed the source code of the SBOM generation tools mentioned previously. This in-depth analysis aims to elucidate the mechanisms through which these tools extract and process dependency information. Our analysis revealed a flaw that compromises the integrity of the generated SBOM data by manipulating the package managers and other files in all tools except CycloneDx for Maven. The generation process is susceptible to manipulation through tampering with package management systems and their associated dependency files. Dependency files such as package.json and requirements.txt specify the software

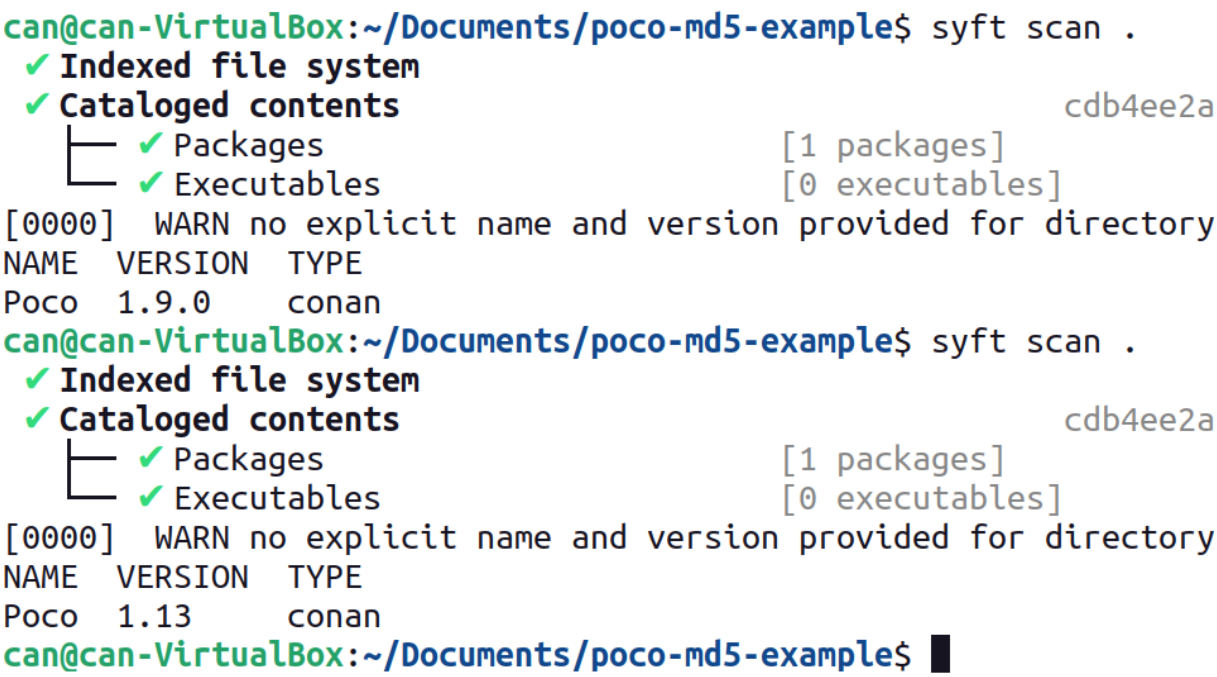


Figure 5.16: Manipulating Generation for C/C++

project's external dependencies and versions. An adversary could introduce misleading information by tampering with these dependency files as well as additional files, which will be discussed in detail in the following paragraphs. This underscores the importance of implementing robust mechanisms within SBOM generation tools beyond solely trusting on dependency and some file information.

The next paragraphs dissect the SBOM generation tools specific to each programming language and systematically elaborate on the experimental results. The results are summarized in Table 5.2.

PHP: It is possible to manipulate the SBOM generation process for PHP projects. Manipulating the contents of the ./composer.json, ./composer.lock, and /vendor/composer/installed.json files in the code base influence the SBOM generation process, thereby potentially compromising the reliability and accuracy of the resulting SBOM artifact. Another interesting outcome is that CycloneDx for PHP does not produce a hash value for components. Since a hash value is not utilized in the consumption phase, as discussed in the Consumption of an SBOM section, it does not break the consumption operation; however, dependency integrity validation cannot be achieved for organizations seeking to achieve it.

Python: It is possible to manipulate the SBOM generation process for projects written in Python. The requirements.txt file is a package management file and dictates the dependencies required for the project's execution. The SBOM generation process can be manipulated by changing the contents of the requirements.txt file in the code base. This could make the final SBOM less reliable and accurate. Another interesting result is that CycloneDx for Python produces no component hash values. Since a hash value is not utilized in the consumption phase, as discussed in the Consumption of an SBOM section, it does not break the consumption operation; however, dependency integrity validation cannot be achieved for organizations seeking to achieve it.

C/C++: The tool used to generate SBOM for C/C++ projects is read-only at the time of writing [71]. The owner archived this repository on Oct 2, 2023. Although there is a newer version of the tool [77], it is not yet a mature project to generate SBOM for C/C++ projects. Hence, we could not analyze the CycloneDX Conan SBOM generation tool. Instead, we used another tool. Syft, supporting C/C++, Debian, Go, and JavaScript ecosystems, among others, was used to generate SBOM for C/C++ projects. It supports the Conan package manager for C/C++ projects, and it is possible to manipulate the SBOM generation process for C/C++ projects utilizing the Conan package manager if an adversary manipulates either conanfile.txt or conanfile.py as can be seen in Fig. 5.16.

Figure 5.17: Different Hash Values in C#

Java: It is not possible to tamper with Java projects that utilize Maven for their build process. There are two tools to generate an SBOM for Java projects: CycloneDX Maven Plugin [72] and CycloneDX Core (Java) [78]. Both are used as Maven plugins. Hence, we focused on the CycloneDX Maven Plugin. The plugin configuration for usage is available in [79]. Maven, a widely used build and package management tool in the Java development ecosystem, has a technique for efficiently managing dependencies. Maven prominently includes a centralized repository that is a reliable and authoritative source for pre-compiled software libraries and dependencies. When a Java project uses Maven, all changes made to the project object model (pom.xml) file swiftly initiate an immediate and efficient updating process. This automated process guarantees that the project includes the designated versions of the dependencies listed in the pom.xml file by retrieving them from the central repository. If an adversary tampers with the version numbers in pom.xml, Maven will download and install the versions the attacker indicated. For this reason, an adversary trying to sabotage the SBOM generation process might actually patch a vulnerability in a dependency on behalf of the company while trying to manipulate the process. Thus, the SBOM generation process for Java projects incorporating Maven for build and dependency management does not allow tampering with the version numbers.

C#: It is possible to manipulate the SBOM generation process for dotnet projects written in C#. The tool used to generate SBOM focuses on dotnet projects written in C#. Changes made to the packages.config file in the code base affect the SBOM generation process, which could make the final SBOM artifact less reliable and accurate. Another significant observation is that the generation tool produces different hash values once you alter version numbers, as Figure Fig. 5.17 shows.

JavaScript: It is possible to manipulate the SBOM generation process for Node.js projects, an open-source, cross-platform JavaScript runtime environment written in JavaScript if adversaries manipulate package.json, package-lock.json, and package.json files under the node_modules folder. Furthermore, we observed that the generation tool generates a hash value for the non-tampered SBOM generation process. In contrast, it does not generate a hash value for any component in a software project, even if only one version number is tampered with. In other words, only one alteration in version numbers is enough to not generate any hash value for all the components in a software project, as seen in Fig. 5.18.

We validated the identified flaw for Javascript as follows. We installed the tool. For demonstration purposes, we once again used the Owasp-developed Juicy Shop application. We generate an SBOM for the Juicy Shop application, summarizing the benign use case of SBOM generation. We then tampered with version values on package.json, package-lock.json, and package.json files under the node_modules folder. Thus, incorrect, tampered SBOM data was generated in the following SBOM generation, posing a security risk for organizations when consumed.

Besides the manipulation of the SBOM generation process, there is another potential security concern regarding the SBOM generation tools. While digital signatures can guarantee the authenticity and integrity of SBOM data, their use remains optional and is not activated by default[43]. This flaw exposes the SBOM generation process

Figure 5.18: No Hash Values for Tampered Dependency in JS

Table 5.2: SBOM Generation Tampering Results

Programming Language	Tampering Possible
Python	yes
C	yes
C++	yes
Java	no
C#	yes
JavaScript	yes
PHP	yes

to manipulation by adversaries.

5.4.5 Conceptual solution

To solve the identified problems in both the SBOM consumption and generations tools, we propose the combination of 3 solutions: (1) secure distribution of SBOMs, ensuring that the SBOMs are transmitted securely from supplier to consumer; (2) secure storage of SBOMs, which protects SBOMs from unauthorized access and tampering; and (3) secure decentralized storage of hash values, facilitating the verification and validation of SBOM authenticity by comparing recorded hashes.

Secure distribution of SBOMs

The distribution of SBOM over insecure channels clearly poses significant security risks. For instance, an SBOM with a library with known vulnerabilities in SBOM could be altered to a version without vulnerabilities. However, this change only happens in the SBOM itself; the actual software remains vulnerable. The SBOM might still seem valid but fail to reflect the software's accurate information. Consequently, attackers could exploit these undisclosed vulnerabilities.

To ensure the secure SBOM distribution between suppliers and consumers, we recommend using secure communication techniques such as Transport Layer Security (TLS) or end-to-end encryption. These mechanisms are crucial for maintaining the authenticity, confidentiality, and integrity of SBOM during distribution. With these safeguards in place, attackers cannot compromise the SBOM during transmission because any tampered files will fail to decrypt correctly.

Secure storage of SBOMs

Secure storage plays an important role in a secure SBOM lifecycle, as its security impacts the entire SBOM lifecycle. To ensure secure SBOM storage at suppliers and consumers, we recommend using digital signatures on

SBOM files and associated software. Upon development or receipt, both parties should sign the SBOM using their private keys with an appropriate validity period for these signatures. If attackers attempt to compromise the SBOM or software to spoof future SBOM consumption, they cannot produce a valid signature without access to the private keys. For enhanced security, a shorter validity period for signatures is required to prevent hackers replacing an SBOM by an outdated version.

Linking SBOMs to the actual software

Secure communication and storage are common security practices, and therefore the two security solutions described above can be relatively easy realized. However, a significant challenge remains: how to link an SBOM to the actual software? This disconnection can occur during SBOM generation, consumption, distribution, and storage. The latter two stages are covered by the solutions above, but security issues persist in SBOM generation and consumption. Currently, SBOM tools use version names instead of hash values to represent software, and therefore there is not a strong connection between the version name and the actual software component (i.e., the code itself). The way forward is to compute the hash of the software and to tie this to the SBOM. However, an important limitation is that there is no practical method for providing a reference repository for hash values of software libraries. Such a reference is vital, because nothing stops an attacker manipulating a software library to adapt the corresponding hash value as well. It is not straightforward to establish such a reference repository for hash values of software libraries. The challenges arise from the complexity of the current software environment. Firstly, the huge amount of software makes it difficult to record hash values for all of them, necessitating an efficient storage mechanism. Secondly, the system is dynamic: software is frequently updated, sometimes even with different authors/owners. Thirdly, the system must handle high throughput and fast response times, as it will receive numerous read and write requests in parallel. Fourthly, some companies may need a private system for software used exclusively within their internal networks. Finally, the solution should be decentralized, as there is no single party which can be trusted on a global scale to manage a reference repository.

The combination of these challenges make the creation of a reference hash table for software a difficult task. Fortunately, there exist at least two large-scale decentralized systems, closely related to our setting, which could be used as a basis to realize our concept. We will briefly discuss these first in the next subsection before zooming in on our proposed solution.

Large-scale decentralized storage solutions

Certificate Transparency The transparency issues associated with software hashes closely resemble those with certificate transparency [80], [81]. The importance of certificate transparency became evident after the DigiNotar hack, which resulted in the issuance of numerous fraudulent certificates [82]. This incident highlighted the need for public records so everyone on a global level can verify certificates, enhancing trust beyond relying solely on Certificate Authorities (CAs). Certificate transparency introduced a system where CAs no longer directly sign certificates for web servers, but instead CAs first send the certificates to log servers. Once logged, the certificates can be signed and issued to the web servers, allowing monitors to review the certificates in the logs and notify web servers if any discrepancies or issues are found. Monitors can be deployed by anyone interested in web security, ensuring the accuracy and integrity of issued certificates. Such a public log system is very suitable in our setting for establishing hash tables for every software being issued, enhancing transparency and allowing for public verification for each software hash.

Blockchain Blockchain was first described in 1991 by Stuart Haber and W. Scott Stornetta to prevent tampering with document timestamps [83]. It functions as a distributed public ledger, consisting of interconnected blocks with a previous hash and proof of work. One of the critical properties of blockchain is its immutability, achieved through the combination of previous hashes and proof of work. The blockchain structure connects all blocks in a sequential chain, where each block includes the previous block's hash. To modify any block, one would need to change all preceding blocks to maintain the integrity of the chain. Proof of work secures the blockchain by requiring significant computational effort to generate a valid hash for each block, meaning that an attacker would need immense computational power to modify a block's hash. The first practical implementation of blockchain was introduced by Satoshi Nakamoto in 2008 with the creation of Bitcoin [84]. Due to its immutability and transparency, blockchain technology has become increasingly popular in various fields, including certificate transparency [85], [86]. Its security properties make blockchain suitable for building a decentralized repository of software hash values. Blockchain's properties eliminate the introduction of third parties and add a layer of security through computational power.

Secure decentralized repository for software hashes: main concept

Inspired by the two decentralized solutions discussed above, we aim to create a public software hash comparison table that serves as a reference for everyone to check and verify software integrity. As described before, certificate transparency also opens possibilities for various fields, including binary transparency for malware fingerprints and ID-to-key mappings [80], which can be used in our design. We propose an SBOM framework with integrity guarantees, as illustrated in Fig. 5.19. This framework involves two groups of entities (software developers and software users) and two decentralized repositories (one for storing the identity information of the software developer/owner and another one for the information on the software – including the hash value). The figure shows the main steps in the process. We will discuss these further in this section.

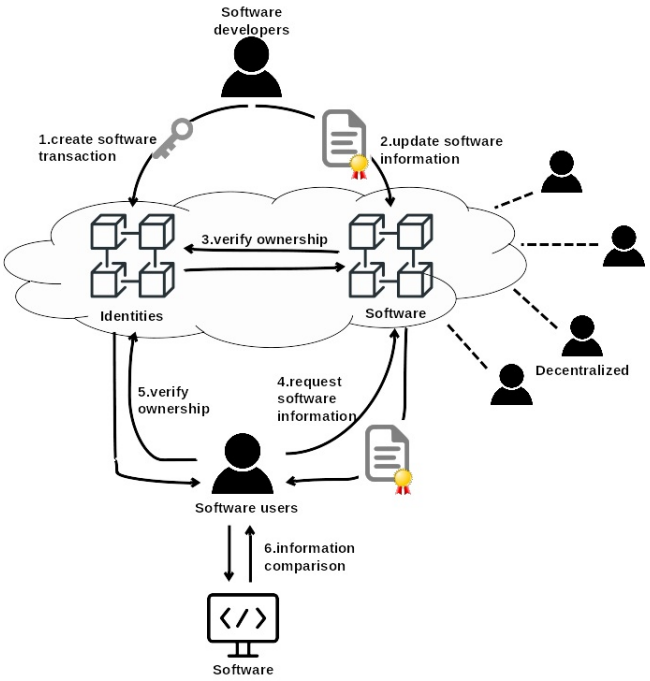


Figure 5.19: Proposed framework for enhanced SBOM integrity

Stakeholders In our framework, there are two primary stakeholders: software developers and software users. Software developers are responsible for developing and uploading the information of their software to the software repository. On the other hand, software users focus on verifying the libraries in software they use. For each library they want to verify, they compute the hash value and compare it against reference data from the software repository. Moreover, both groups can serve as decentralized contributors to the identity and software repository.

Repositories In our proposal, there are two repositories: the identities repository and the software repository. The identities repository serves as access control for the software repository. Developers can modify software information in the software repository only if they can prove ownership by relying on the identities repository. On the other hand, the software repository records software details, such as the software name and its hash value. This setup allows software users to verify the integrity of any software by comparing their software hash with the hash stored in the software repository.

Enhanced SBOM integrity by decentralized public repositories As illustrated in Fig. 5.19, our framework is divided into two main parts: a registration phase and a verification phase. Lets now zoom in on the different steps: (1) To register software, a developer must generate their public/ private key pairs and register their ownership of this software in the identities repository. This is only possible if the software name does not exist; otherwise, software ownership can only be transferred from the current owner (see further in this section). (2) Once ownership is confirmed, the developer uploads the software's details, such as its hash value, to the software repository. (3) The software repository then checks with the identities repository to confirm the developer's ownership. If confirmed, the information is successfully uploaded; if not, the upload fails. Additionally, it is important to increment the software version for each new version that gets registered. (4) If users want

to verify the validity of the software, for example during SBOM generation or consumption, users will request reference information from the software repository. Upon receiving the reference information, users check the transaction's signature using the owner's public key. (5) If the signature is verified, the identities repository is then consulted to confirm the owner's ownership of the software when the information was recorded in the software repository based on the owner's public key and the timestamp. (6) Once the identities repository confirms ownership, users can trust the reference information provided by the software repository to validate the software they intend to use.

Ownership transfer in identities repository Ownership transfer occurs when a software owner decides to pass software ownership to another party, which happens in the identities repository. In our design, we consider a software name as a property, and transferring ownership means starting a transaction where the current owner assigns the software name to the new owner. Once recorded, the software repository can verify the new ownership details from the identities repository. However, the new owner must verify their software ownership from the identities repository before adding any data to the software repository to prevent security issues.

5.4.6 Discussion

In the previous section, we outlined our main concept for the decentralized storage and verification of reference information (including the hash value) of software libraries. This section will zoom in more on some practical considering regarding the realization of such a system, including extensions for private systems. Moreover, we will also discuss the feasibility of different implementation approaches.

Implementation Approaches

Blockchain-based solution Due to the decentralized nature of the system, a blockchain-based solution is an obvious choice. There are two main repositories in our design: an identities repository and a software repository. The use of blockchain gives us integrity guaranties and supports transaction-based ownership control. The identities repository guarantees a developer's identity and their ownership of specific software. Unlike a traditional blockchain that records cryptocurrency transactions, this identity blockchain will record the software transactions, tracking software creation, transfer, and ownership. For instance, if developer A is the first to declare he created software S, the identities blockchain will log this, preventing other developers from claiming ownership of software S. Ownership can only change if developer A creates a transaction to transfer it to another developer, such as developer B. The software repository stores key details about the software, such as hash values, creation time, and other identity information. Developers who own their software can upload this information to the blockchain as a reference for software users. The blockchain will deny the upload if a developer does not own the software. Additionally, software users can verify a software library by comparing it with the reference information stored on the blockchain. Each transaction in both the identities and software repositories is designed to include at least the following elements:

For transfer of ownership:

1. *Unique transaction id: A distinct identifier of the transaction.*
2. *Timestamp: The time when the transaction was initiated.*
3. *Input public key: The public key of the current software owner.*
4. *Output public key: The public key of the future software owner.*
5. *Transactions: The specific software ownership.*
6. *Signature: A signature created with the input's private key.*

For registering software in the repository:

1. *Unique transaction id: A distinct identifier of the transaction.*
2. *Timestamp: The time when the transaction was initiated.*
3. *Owner public key: The public key of the software owner.*
4. *Software name: Unique name of the software.*
5. *Version: The version of the software.*

6. *Hash value: Hash of this version of the software.*
7. *Optional: Some optional information.*
8. *Signature: A signature created with the owner's private key.*

Certificate transparency-based solution An alternative strategy could be to reuse the concepts from certificate transparency, including the use of monitors to ensure the security of log servers. In our design, all stakeholders can act as monitors. Software developers should monitor the identities repository to verify the correctness of their identities, such as public keys. Additionally, they should monitor the software repository to detect any incorrect or unauthorized software information. On the other hand, software users should monitor for any changes in the previously immutable software information, as the repository should be append-only and previous software information should remain unchanged.

Private repositories for reference hash values

Some companies require private repositories for their software information due to privacy concerns. Motivated by certificate transparency, we proposed a solution tailored for private repositories, shown in Fig. 5.20. For private repositories, the number of participants is small and access control is managed internally. Therefore, using blockchain to eliminate third-party access control is unnecessary. For private repositories, we proposed traditional table structures and Merkle trees, which are more efficient and computationally friendly. In addition, they avoid the need for proof of work, reducing vulnerability to attacks from entities with substantial computational power. As illustrated in Fig. 5.20, the framework includes two groups (software developers and software users) and two repositories (one for access control and another for software information), which are similar to the public system. As outlined in the figure, our framework involves two main parts: registering and verifying software information via logs. (1) To upload the software information, software developers are first verified by the access control system. Developers may apply for permissions in advance or receive them automatically based on company policies. The access control then checks if the developers have the appropriate permissions for the specified software. (2) Once verified, developers are allowed to upload the software information to the logging server. (3) Software users who wish to verify software must communicate with the access control entity to confirm their permissions. (4) After confirming their permissions, they can access software information from the logs.

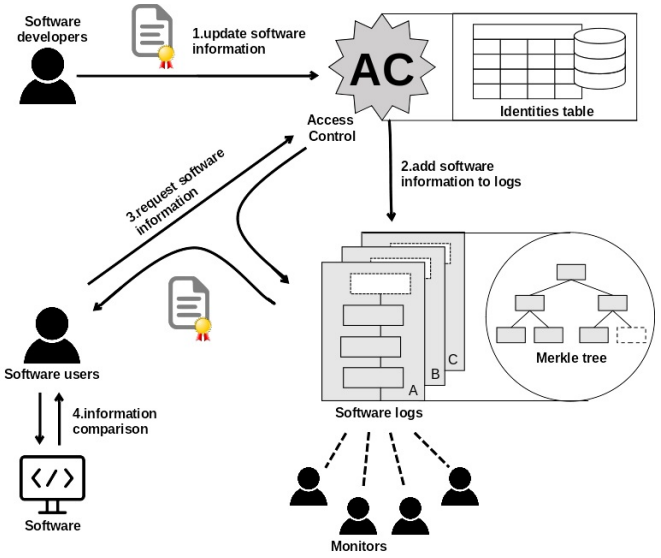


Figure 5.20: Proposed framework for enhanced SBOM integrity with Access Control

To log software information, we use Merkle tree-based repositories. A Merkle tree is a binary tree structure consisting of numerous nodes, with the top-most node known as the root and the bottom-most as leaf nodes. Software information is stored in the leaf nodes, and all other nodes, except for the leaves, contain hash values derived from the nodes below them. For instance, each parent node has two child nodes and stores a hash value generated from these children. This setup ensures that every node, except the leaves, represents a summary of all nodes beneath it. In this case, the root node summarizes all leaf nodes. Any modification to the software

information will alter the root hash, enabling users to efficiently verify the software's integrity by comparing multiple root hashes. The repository record includes:

1. *Unique record ID: Unique identifier for this record.*
2. *User ID: Unique identifier for each user.*
3. *Software name: Unique name of the software.*
4. *Version: The version of software.*
5. *Hash value: Hash of software at this version.*
6. *Optional: Some optional information.*
7. *Timestamp: The time when this record was initiated.*

Feasibility

We argue that it is feasible to deploy a decentralized solution for storing and verifying reference information of software libraries, including their hash values. As a reference, we will refer to Bitcoin and Certificate Transparency; both have been deployed on a global scale.

Table 5.3: The Size of Data Elements in Identity and Software Repository

Item	Size (byte)
Identity Repository	
ID (SHA-256)	32
Timestamp	8
Input Public Key (ECDSA-256)	32
Output Public Key (ECDSA-256)	32
Transactions Content	128
Signature (ECDSA)	64
Total	296
Software Repository	
ID (SHA-256)	32
Timestamp	8
Owner Public Key (ECDSA-256)	32
Software Name	32
Version Number	8
Hash Value	32
Signature (ECDSA)	64
Total	208

Storage For our decentralized public repository, the size of each transaction are shown in Table 5.3, where the total size of identities and software repositories are 296 and 208 bytes, respectively. In 2023, Github reached 100 million developers and 284 million public repositories [87]. We assume that the average number of versions of each software is 10. Thus, using these numbers one gets a total size for a full node of the identities blockchain of 27.57 GB, and 550.15 GB for the software blockchain. It should be mentioned that some public repositories on Github are documentation or resources instead of software, meaning that the actual total size will be smaller. In comparison, the total size for a full node in Bitcoin was 562.61 GB on 21st May 2024 [88]. Therefore, we can conclude that the storage requirements for our framework (27.57 GB and 550.15 GB) are manageable. For a decentralized private repository, storage requirements are expected to be significantly lower.

Verification Speed Verification speed refers to how quickly a software user can verify all the hashes of the software entries in an SBOM using blockchain. This process is similar to the "check a transaction and the certificate" speed described in [85], which is 0.25 ms per verification. Based on this, we estimate a verification time of 0.5 ms and 1.5 ms for identity and software blocks, respectively, considering the 100 million users and 300 million public repositories. According to the npm official website in 2019, the average modern web application has over 1000 dependencies. Thus, the estimated total verification time for a user is (0.5 + 1.5) × 1000 ms = 2 seconds, which includes software hash verification in software blockchain and ownership verification in the

identities blockchain [89]. An SBOM verification process of 2 seconds is feasible, and further improvements could be achieved by caching frequently verified software hashes. Similarly as for storage, there are no speed concerns for the private repository, as it does not rely on blockchain techniques and has significantly fewer participants and software.

Registration Speed Registration speed refers to how quickly blocks are created in the identities and software blockchain. To evaluate the registration speed, we collected the average confirmation time of Bitcoin over three years, from June 2021 to June 2024 [90]. The maximum, minimum, median, and average time were 25809, 5, 40, and 262 minutes, respectively. To avoid the influence of the outliers, such as 25809 minutes, we used the median value of 40 minutes as the expected confirmation time for our decentralized public repository system, which is an acceptable speed for a global reference system for software libraries.

5.4.7 Related Work

This study is the first to comprehensively explore both the theoretical and practical aspects of generating and consuming SBOMs to increase the supply-chain security.

A notable study has introduced a blockchain framework designed to improve SBOM sharing, featuring an authoritative structure and a tree-based certificate system. This framework capitalizes on the inherent integrity properties of blockchain technology, offering a potential solution for SBOM sharing challenges [91]. However, issues remain with linking SBOMs to actual code, generating SBOMs with correct software hashes, and correctly consuming SBOMs. MITRE ATT&CK proposed an end-to-end framework for software supply chain integrity, inspired by the SolarWinds incident. It emphasizes standardizing SBOMs, improving signature infrastructure, and enhancing automation, integration, and cryptographic traceability. Specifically, it advocates for strengthening the generation and distribution process of SBOMs, highlighting their critical importance [92]. L.J. Camp et al. assess the risks associated with SBOMs and the immediate need for SBOM standardization. The authors argue that without widespread adoption of robust protocol security, SBOMs risk failing to meet their security promise [93]. Although both works emphasize the importance of SBOM standardization and distribution, significant improvements in actual implementation are still required. In conclusion, current studies mainly focus on SBOM standardization and distribution, neglecting the generation and consumption parts, e.g., a global software comparison table for SBOM.

In addition to SBOMs, various traditional methods, e.g., distribution of hash, threat modelling, software composition analysis, are used in supply-chain security. MITRE ATT&CK advises verifying complied code binaries against known good hashes and using other integrity mechanisms to protect against software supply chain injections. They suggest distributing the software's cryptographic hash through independent, trusted mechanisms accompanied by code signing [94]. The idea of comparing software hashes is similar to SBOM; however, it lacks a method for storing these known-good hashes. Our proposal also offers a secure framework for storing these known-good hashes. Jeong Yang et al. highlighted four key mechanisms to detect or prevent software supply chain attacks: (1) Github's Software Composition Analysis (SCA) tools that identify and patch vulnerabilities in open-source dependencies; (2) CISA's CHIRP (CISA Hunt and Incident Response Program), which utilizes Indicators of Compromise (IoCs) to detect threats after a compromise; (3) SootDiff, a tool that identifies code clones produced by different compliers using bytecode clone detection techniques; (4) The in-toto framework, which cryptographically secures the software supply chain and enables end-users to verify all steps within the chain [95]. These mechanisms are similar to SBOM and will be integrated with or replaced by SBOM with SBOM development. Guanyi Lu et al. divided supply chain security into four categories: detection, prevention, response, and mitigation. They operationalized these categories using various indicators to assess and explain the performance of supply chain security across 462 firms in the United States and Italy [96]. Naeem Firdous Syed et al. performed a STRIDE threat modeling analysis on SBOM traceability data, organizing it into four layers: application, data sharing, data capture, and data carrier. The authors describe common threats within these layers and offer countermeasures, helping stakeholders understand and mitigate potential cybersecurity risks and vulnerabilities [97]. Although threat modeling provides a general security strategy, our proposal offers a detailed implementation for SBOM integrity in supply chain security. S.E. Chang et al. [98] and Pankaj Dutta et al. [99] reviewed 106 and 178 research articles, respectively, on blockchain implementation in supply chains. Both studies enhance understanding of blockchain applications in supply chain management and offer a blueprint based on literature analysis. Besides exploring blockchain applications, Sana Al-Farsi et al. identified several challenges that impede the reliability and security of blockchain-based supply chain management (BC-SCM) systems, helping to establish the desired security of the BC-SCM [100]. Nevertheless, these works do not address blockchain applications for SBOM integrity, and our study extends blockchain applications to

SBOM integrity.

With the excellent performance of artificial intelligence (AI), it is increasingly being used to enhance supply chain security. Malka N. Halgamuge et al. explore state-of-the-art deep learning techniques to secure supply chains in the renewable energy industry. The authors highlight key model design factors for combating adversarial threats and detail various deep learning architectures, assessing their strengths and limitations in security applications [101]. People also use various AI models to predict defects in software, enhancing supply chain security. These models are trained on large datasets of defective and defect-free code to predict software defects, enabling the detection of issues like outdated or malicious packages. Elena N. Akimova et al. review deep learning-based techniques for software defect prediction, discussing current challenges and future directions in this field [102]. Szymon Stradowski et al. focused on predicting software defects in industrial applications through machine learning. They review studies from 2015 to 2022, discussing the AI models, datasets, and costs associated with software defect prediction in industry [103]. Most AI studies focus on directly predicting defects from code, which can be too time-consuming and complex for consumers to verify vulnerabilities in large applications. In contrast, SBOM consumption is more straightforward.

Instead of using AI in defect detection, people have started to apply AI in SBOM. B. Xia et al. propose "AIBOMs," an adaptation where traditional SBOMs, which list software components, are extended to suit AI systems' dynamic and evolving nature. AIBOMs specifically address the challenges posed by AI systems like reinforcement learning, which continuously evolves with new data, by creating a tailored BOM for these dynamic environments [91]. Petar Radanliev et al. highlighted that technologies such as machine learning, including generative pre-trained transformers and natural language processing, can automate processes in the SBOM consumption with Vulnerability-Exploitability eXchange (VEX) [104]. Heeyeon Kim et al. developed a binary classifier to identify defective software before SBOM generation. By removing these defective software, the authors can produce an SBOM consisting solely of defect-free software [105]. These mechanisms add a security layer before SBOM generation, while they do not address the issue of linking SBOMs to the actual code.

5.4.8 Conclusion

Our work revealed several significant security weaknesses in the processes and tools for SBOM consumption and generation currently widely used in practice. For SBOM consumption, all the investigated tools, including the OWASP dependency tracker, only rely on basic checks of library names and versions and omit the verification of hash values. For SBOM generation, we investigated the SBOM generation process for the seven most common programming languages and have shown that for all languages except one, the SBOM file is generated merely based on library configuration files, hence disregarding the actual code. Moreover, we discovered that none of the tools enabled the signing of an SBOM by default during generation nor required SBOMs to be signed during consumption. To mitigate these issues, we propose a a mitigation strategy that includes the decentralized storage of hash tables, either public or private, that explicitly links each entry in an SBOM to its corresponding software library.

5.4.9 Appendix: Data Availability

In this work, we generated data for both SBOM consumption and SBOM generation. For the SBOM consumption part, we utilized two versions of the Juicy Shop application: a clean version and a manipulated version. The clean Juicy Shop SBOM is provided by CycloneDX, which is available at CycloneDX GitHub Repository (https://github.com/CycloneDX/bom-examples/blob/master/SBOM/juice-shop/v11.1.2/bom.json). Our manipulated Juicy Shop SBOM will be publicly released in a dedicated public repository. For the SBOM generation part, we examined SBOM generation across multiple programming languages. The studied source codes, manipulated files and the related SBOMs will be made available in a public repository.

5.5 Machine Learning-Based Device Identification for Modbus RTU RS-485 Systems (Xinhai Zou, Dave Singelée)

5.5.1 Introduction

In industry, the increasing interconnection of Internet-of-Things (IoT) devices enhances operational efficiency. However, it also raises security risks, such as device spoofing. This study investigates the application of machine learning for device identification in Modbus systems based on voltage characteristics to reduce the likelihood of

spoofing attacks. By analyzing the voltage data from a simulated Modbus RTU RS-485 setup with one master and nine slaves, the proposed method achieves an accuracy of 98.39% and reduces the spoofing success rate to 7.17%.

5.5.2 Methodology

Overview of Methodology

The work starts with the Circuit Design and Data Collection, which involve building the Modbus system and collecting data for the machine learning analysis. The Machine Learning step applies techniques for device identification based on physical-layer characteristics of the Modbus devices. Finally, Result Analysis interprets the outcomes.

Modbus RTU RS-485 Protocol

Modbus RTU RS-485 Protocol combines the Modbus RTU protocol with the RS-485 serial transmission standard, both widely used in industrial applications. A typical Modbus RTU RS-485 system, illustrated in Fig. 5.21, consists of a master device and up to 31 slave devices. Communication between master and slaves occurs via differential signaling on A and B cables, where A > B represents a binary 1, and A < B represents a binary 0. It is a half-duplex system, which only allows one transmitter (master or slave) to be active at any given time.

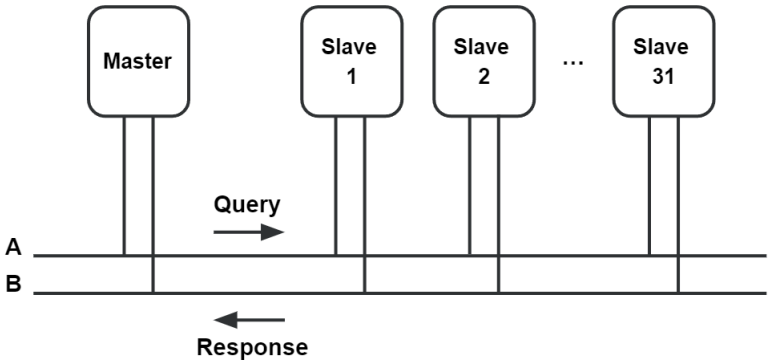


Figure 5.21: Modbus RTU RS-485 system

Machine Learning Model Design

In the Machine Learning phase, both random forest and neural network algorithms are applied. Random Forest is an ensemble learning method that combines multiple decision trees, each built using entropy-based splitting criteria, as shown in Fig. 5.22(a). Neural Network utilizes a loss function to iteratively adjust pre-defined model parameters, allowing the model to approximate the target values closely, as illustrated in Fig. 5.22(b).

5.5.3 Implementation

Implementation of Modbus RTU RS-485 System

This experiment constructs a Modbus RTU RS-485 lab environment comprising one master, nine slave devices, and one attacker's slave. Each master and slave is built using an Arduino UNO R4 Minima and a RS485-TTL converter module, as illustrated in Fig. 5.23. Arduino UNO R4 Minima and RS485-TTL module are powered by 5V USB ports and connected to the same ground. All master and slaves are connected in parallel with jumper wires serving as the A and B cables for RS-485 communication, as shown in Fig. 5.24. An NI USB-6351 voltage sensor is also integrated into the system to measure physical characteristics during experiments. In addition to the benign environment set-up, an attack scenario is made by replacing the master or one of the slaves with an attacker's device. The attacker's device, identical to the original master or slave units, is constructed using an Arduino UNO R4 Minima and an RS485-TTL converter module.

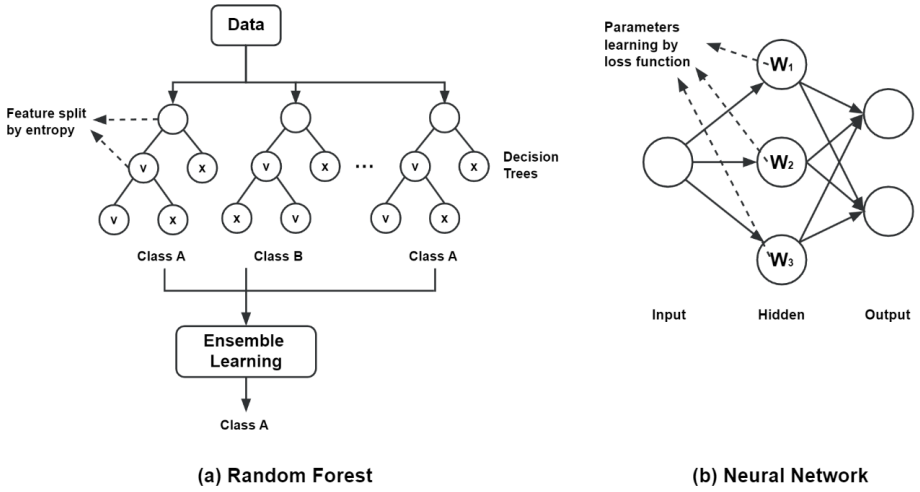


Figure 5.22: Model structure of (a) Random Forest and (b) Neural Network

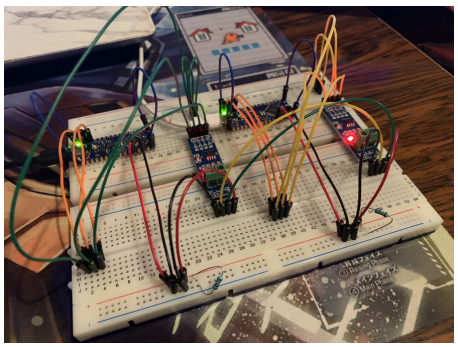


Figure 5.23: Circuit of a Modbus RTU RS-485 Master/Slave

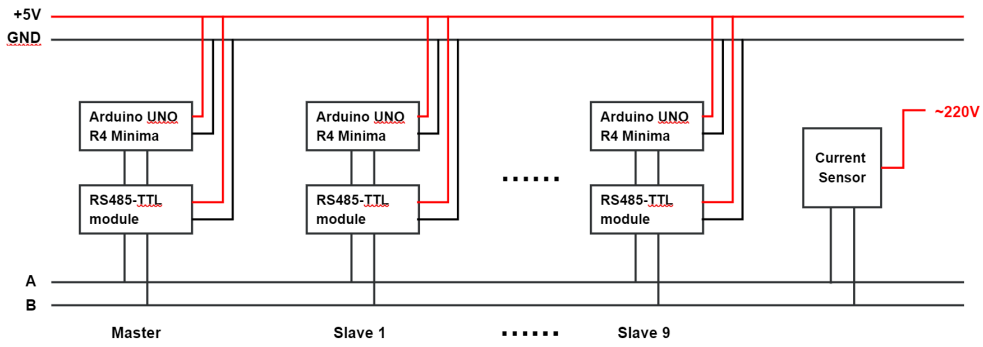


Figure 5.24: Circuit of a Modbus RTU RS-485 Master/Slave

Data Collection

In the Modbus protocol, the master can read from and write messages to the slaves. For each slave, we define four reading and three writing messages, as shown in Table 5.4. Each message is measured for five minutes. Each register has an initial value listed as "Value on Register" in the table. For a writing message, it has a value called "Value after Write." For example, the message "$< ID >, 0, < ID > 0, < ID >< ID >$" indicates that the initial value of register 0 on slave $< ID >$ is "$< ID > 0$", and it is a writing message where the value after writing will be "$< ID >< ID >$". All four reading and three writing messages are collected for each of the nine slaves. In the attack scenario, the attacker's device iteratively replaces each slave, from slave 0 to slave 9.

Table 5.4: Reading and Writing Message during experiments

	Slave ID	Register ID	Value on Register	Value after Write
Reading	<ID>	0	<ID>0	-
	<ID>	1	<ID>1	-
	<ID>	2	88	-
	<ID>	3	99	-
Writing	<ID>	0	<ID>0	<ID><ID>
	<ID>	2	88	55
	<ID>	4	1	1

Machine Learning Set-up

Data preprocessing Voltage data from different devices is scaled to the range from 0 to 1 using a MinMaxScaler. Ten features are extracted from the original voltage data, as outlined in Table 5.5. Voltage data from devices is segmented into data slices with a specified window length. After comparing different window lengths on a random forest classifier, shown in Fig. 5.25, a window length of 45 obtains the best results, which is used in subsequent Machine Learning analyses. The total data of training and testing dataset is 15389 and 3842, and the ratio of training and testing dataset is 8:2.

Table 5.5: Ten features for Machine Learning

	Features
Voltage 1 (Cable A)	Mean Standard Deviation Minimum Value Maximum Value Mean Magnitude of FFT
Voltage 2 (Cable B)	Mean Standard Deviation Minimum Value Maximum Value Mean Magnitude of FFT

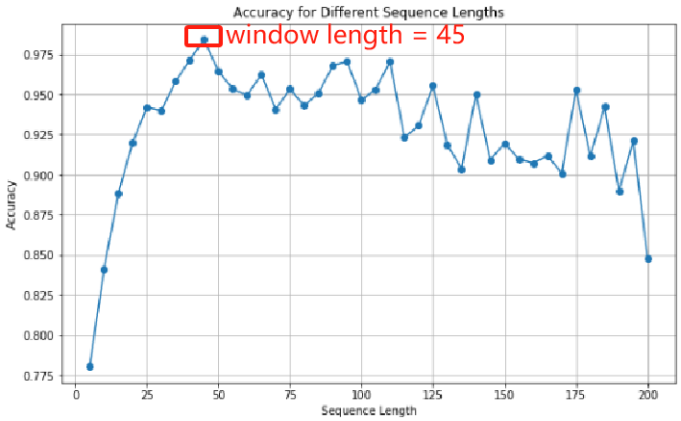


Figure 5.25: Accuracy for different window length

Machine Learning Techniques In this experiment, the RandomForestClassifier from the scikit-learn package is used to extract device fingerprints from simulated Modbus master and slave devices. While the neural network discussed in the previous section is not applied at this stage, it will be used for more complex tasks in future steps.

5.5.4 Discussion

Random Forest

The random forest achieved an accuracy of 98.39% on the voltage dataset based on ten selected features. In the attack scenario, the spoofing success rate is 7.17%, indicating that the attacker can only have a 7.17% chance to impersonate a slave or master successfully. The feature importance is studied to evaluate and potentially improve the model, as shown in Fig. 5.26. Voltage 1 (Cable A) is found to be more influential than Voltage 2 (Cable B). The top five important features are the maximum value of Voltage 1, the minimum value of Voltage 1, the mean of Voltage 2, the minimum value of Voltage 2, and the mean of Voltage 1.

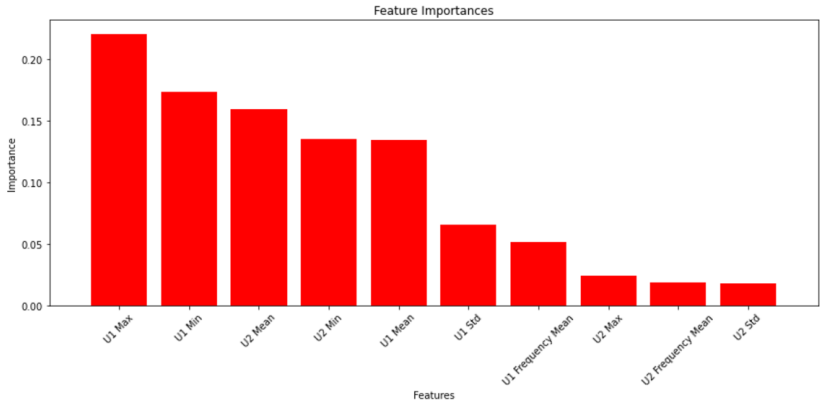


Figure 5.26: Features Importance of results from Random Forest Model

As shown in Fig. 5.27, some devices (1, 3, 4, and 5) exhibit continuously high maximum values, while other devices (2, 6, 7, 8, and 9) display lower maximum values. This variation may be influenced by the physical position of each slave device; for instance, devices 1 through 5 are positioned closer to the device 0 (master) than devices 6 through 9. This positional effect may lead the model to learn the spatial relationships rather than device fingerprints. To mitigate this issue and enhance model generalization, further study in varying the positions of each device is necessary.

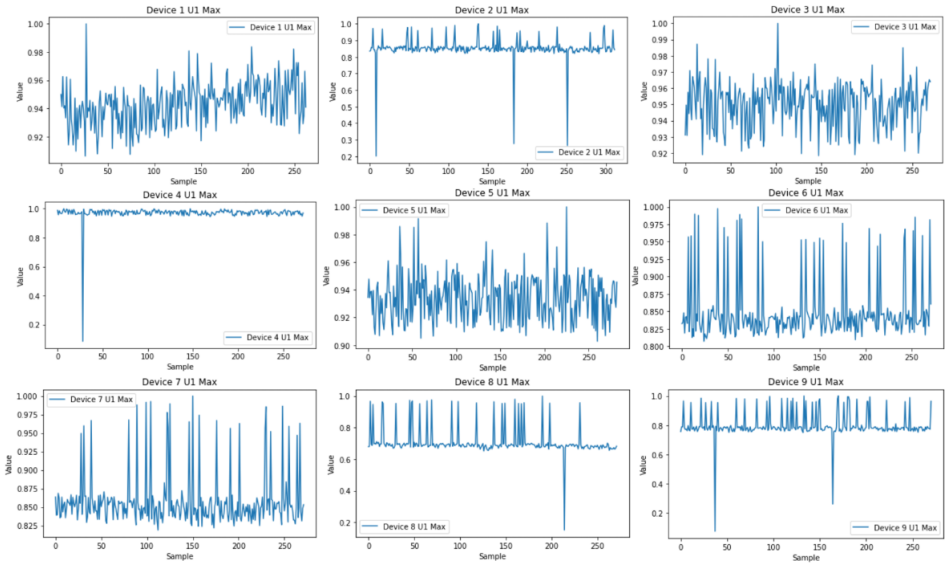


Figure 5.27: Maximum value of Voltage 1 for all slaves

5.5.5 Conclusion

This experiment achieves an accuracy of 98.39% in a simulated Modbus RTU RS-485 system with one master and nine slaves. The spoofing success rate for an attacker posing as a legitimate slave was only 7.17%. From the study of feature importance, the highest impact from voltage maximum value may indicate that the model may only learn device positions instead of actual device fingerprints. Therefore, future work should explore varying device positions to mitigate this effect. Furthermore, a more complex and realistic experimental setup would be valuable as a set of only 10 devices is limited for real-world applications.

6

Conclusion

6.1 Main achievements

The deliverable presents a cohesive set of outcomes that collectively advance the state of cybersecurity in cyber-physical systems. For cyber-space observability, the DxAgent framework and SMOOTHIE load-balancing techniques deliver real-time operational insights and enhanced network performance. IT anomaly detection leverages deep learning to identify malicious activity even in encrypted traffic, achieving unprecedented accuracy. Cyber-operator characterization emphasizes the importance of human reliability in mitigating cascading failures, while scenario-based analysis highlights the critical role of manual corrective actions. In the realm of cyber-attack resilience, decentralized authentication protocols, system polymorphism, and improved supply chain security frameworks address key challenges, offering practical solutions to enhance robustness. Collectively, these contributions strengthen the overall security posture and ensure resilience against evolving cyber threats.

6.2 Lessons learned

Throughout the work on Task 3.2, one key lesson was the necessity of adopting a holistic approach to securing cyber-physical systems. The complexity of modern energy systems necessitates safeguarding every component, from the cyber-infrastructure to physical equipment. Ensuring consistent communication integrity, robust anomaly detection, and effective recovery strategies are essential, as threats often exploit the interdependencies between components. Furthermore, human operators remain a crucial factor in maintaining system stability, highlighting the need for comprehensive training and reliable decision-support tools.

Another critical insight was the value of balancing preventive measures with corrective responses. While proactive strategies are essential to reduce vulnerabilities, adaptive real-time decision-making during incidents ensures resilience against unforeseen threats. This balance can only be achieved through collaboration between multidisciplinary teams spanning IT, electrical engineering, and operations.

6.3 Needs for further research

The work conducted thus far represents significant progress, but several areas remain ripe for further exploration. Firstly, the integration of cyber and electrical layers in real-time analysis and response mechanisms is underutilized. Leveraging the interplay between these layers could enable innovative solutions for both attack detection and dynamic recovery strategies. For instance, scenarios where malicious exploitation of ICT systems leads to widespread effects on the electrical grid—such as an attack instructing all electric vehicles to simultaneously charge, causing grid instability—highlight the need for coordinated mitigation strategies. Developing mechanisms to prevent such actions by refusing or delaying charging requests during critical periods could enhance grid resilience.

Additionally, anomaly detection systems could be tailored to identify unusual communication patterns from connected devices, such as electric vehicles. If a device like a Tesla begins transmitting messages unrelated to charging, this could signal a potential cyber-attack or system misconfiguration, prompting immediate intervention. These insights reinforce the necessity of bridging the gap between electrical and ICT domains, creating unified frameworks capable of addressing cross-domain risks effectively.

Finally, the development of predictive tools capable of simulating complex attack scenarios and their potential impact on both cyber and physical systems is a promising avenue. Such tools could enhance preemptive strategies and provide actionable insights for system operators. As the threat landscape evolves, new challenges such as the integration of decentralized energy resources and increased reliance on IoT devices will require tailored security solutions. This underscores the importance of continuing interdisciplinary research to address emerging vulnerabilities and ensure the long-term resilience of cyber-physical systems.

Bibliography

[1] J. Runge, A. Gerhardus, G. Varando, V. Eyring, and G. Camps-Valls, "Causal inference for time series," *Nature Reviews Earth & Environment*, vol. 4, no. 7, pp. 487–505, Jul. 2023, ISSN: 2662-138X. DOI: 10.1038/s43017-023-00431-y. [Online]. Available: https://doi.org/10.1038/s43017-023-00431-y.

[2] A. Gerhardus and J. Runge, "High-recall causal discovery for autocorrelated time series with latent confounders," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 12 615–12 625. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/94e70705efae423efda1088614128d0b-Paper.pdf.

[3] L. Champagne and B. Donnet, "Smoothie: Efficient and flexible load-balancing in data center," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, 2024, pp. 1–7. DOI: 10.1109/NOMS59830.2024.10575467.

[4] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1094–1104, Oct. 2001, ISSN: 1045-9219. DOI: 10.1109/71.963420. [Online]. Available: https://doi.org/10.1109/71.963420.

[5] N. Katta, A. Ghag, M. Hira, *et al.*, "Clove: Congestion-aware load balancing at the virtual edge," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '17, Incheon, Republic of Korea: Association for Computing Machinery, 2017, pp. 323–335, ISBN: 9781450354226. DOI: 10.1145/3143361.3143401. [Online]. Available: https://doi.org/10.1145/3143361.3143401.

[6] J. E. Sullivan and D. Kamensky, "How cyber-attacks in ukraine show the vulnerability of the u.s. power grid," *The Electricity Journal*, vol. 30, no. 3, pp. 30–35, 2017, ISSN: 1040-6190. DOI: https://doi.org/10.1016/j.tej.2017.02.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1040619017300507.

[7] R. Deng, G. Xiao, R. Lu, H. Liang, and A. V. Vasilakos, "False data injection on state estimation in power systems—attacks, impacts, and defense: A survey," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 411–423, 2017. DOI: 10.1109/TII.2016.2614396.

[8] D. U. Case, "Analysis of the cyber attack on the ukrainian power grid," *Electricity information sharing and analysis center (E-ISAC)*, vol. 388, no. 1-29, p. 3, 2016.

[9] I. Cisco Systems, *Encrypted traffic analytics*, https://www.cisco.com/c/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf, May 2019.

[10] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related features," in *Proceedings of the 2nd International Conference on Information Systems Security and Privacy - ICISSP*, INSTICC, SciTePress, 2016, pp. 407–414, ISBN: 978-989-758-167-0. DOI: 10.5220/0005740704070414.

[11] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[12] R.-H. Hwang, M.-C. Peng, V.-L. Nguyen, and Y.-L. Chang, "An lstm-based deep learning approach for classifying malicious traffic at the packet level," *Applied Sciences*, vol. 9, no. 16, 2019, ISSN: 2076-3417. DOI: 10.3390/app9163414. [Online]. Available: https://www.mdpi.com/2076-3417/9/16/3414.

[13] G. Baldini, "Analysis of encrypted traffic with time-based features and time frequency analysis," in *2020 Global Internet of Things Summit (GIoTS)*, 2020, pp. 1–5. DOI: 10.1109/GIOTS49054.2020.9119528.

[14] Z. Tang, J. Wang, B. Yuan, H. Li, J. Zhang, and H. Wang, "Markov-gan: Markov image enhancement method for malicious encrypted traffic classification," *IET Information Security*, vol. 16, no. 6, pp. 442–458, 2022. DOI: https://doi.org/10.1049/ise2.12071. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/ise2.12071. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/ise2.12071.

[15] M. Vaiman *et al.*, "Mitigation and prevention of cascading outages: Methodologies and practical applications," in *2013 IEEE Power & Energy Society General Meeting*, IEEE, 2013, pp. 1–5.

[16] X. Zhang, C. Zhan, and K. T. Chi, "Modeling the dynamics of cascading failures in power systems," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 7, no. 2, pp. 192–204, 2017.

[17] Z. Wang *et al.*, "Impacts of operators' behavior on reliability of power grids during cascading failures," *IEEE Transactions on Power Systems*, vol. 33, no. 6, pp. 6013–6024, 2018.

[18] M. Zakariya and J. Teh, "A systematic review on cascading failures models in renewable power systems with dynamics perspective and protections modeling," *Electric Power Systems Research*, vol. 214, p. 108 928, 2023.

[19] S. Kamyab, P.-E. Labeau, and P. Henneaux, "Modeling manual corrective actions in probabilistic risk assessment of cascading outages," *Electric Power Systems Research*, vol. 234, p. 110 831, Sep. 2024. DOI: https://doi.org/10.1016/j.epsr.2024.110831.

[20] P. H. and P.-E. L. S. Kamyab, "Evaluation of risk of cascading outages due to imperfect manual corrective actions," in *IEEE PES ISGT EUROPE*, Dubrovnik, Croatia: IEEE, 2024.

[21] M. Panteli, P. A. Crossley, D. S. Kirschen, and D. J. Sobajic, "Assessing the impact of insufficient situation awareness on power system operation," *IEEE Transactions on Power Systems*, vol. 28, no. 3, pp. 2967–2977, 2013.

[22] NIST, "Nist special publication 1500-201: Framework for cyber-physical systems, working group reports," National Institute of Standards and Technology, Tech. Rep., 2017.

[23] G. Liang, J. Zhao, F. Luo, S. R. Weller, and Z. Y. Dong, "A review of false data injection attacks against modern power systems," *IEEE Transactions on Smart Grid*, vol. 8, no. 4, pp. 1630–1638, 2016.

[24] S. Institute, "Sans 504: Hacker tools, techniques, exploits, and incident handling," Bethesda, MD, USA, Tech. Rep., 2024.

[25] A. Von Meier, *Electric power systems: a conceptual introduction*. John Wiley & Sons, 2006, ch. 6.

[26] R. H. Lasseter and P. Paigi, "Microgrid: A conceptual solution," in *2004 IEEE 35th annual power electronics specialists conference (IEEE Cat. No. 04CH37551)*, IEEE, vol. 6, 2004, pp. 4285–4290.

[27] F. S. Al-Ismail, "Dc microgrid planning, operation, and control: A comprehensive review," *IEEE Access*, vol. 9, pp. 36 154–36 172, 2021.

[28] C. Li, C. Cao, Y. Cao, Y. Kuang, L. Zeng, and B. Fang, "A review of islanding detection methods for microgrid," *Renewable and Sustainable Energy Reviews*, vol. 35, pp. 211–220, 2014.

[29] H. Han, X. Hou, J. Yang, J. Wu, M. Su, and J. M. Guerrero, "Review of power sharing control strategies for islanding operation of ac microgrids," *IEEE Transactions on Smart Grid*, vol. 7, no. 1, pp. 200–215, 2015.

[30] L. Lu and J. Lu, "A lightweight verifiable secret sharing in internet of things," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 5, 2022.

[31] L. Harn, "Group authentication," *IEEE Transactions on computers*, vol. 62, no. 9, pp. 1893–1898, 2012.

[32] M. A. Hossain, H. R. Pota, S. Squartini, F. Zaman, and K. M. Muttaqi, "Energy management of community microgrids considering degradation cost of battery," *Journal of Energy Storage*, vol. 22, pp. 257–269, 2019.

[33] J. Zheng, D. W. Gao, and L. Lin, "Smart meters in smart grid: An overview," in *2013 IEEE Green Technologies Conference (GreenTech)*, 2013, pp. 57–64. DOI: 10.1109/GreenTech.2013.17.

[34] K. Nyberg, "Fast accumulated hashing," in *International Workshop on Fast Software Encryption*, Springer, 1996, pp. 83–87.

[35] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[36] M. Stadler, "Publicly verifiable secret sharing," in *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 1996, pp. 190–199.

[37] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," in *Concurrency: the works of leslie lamport*, 2019, pp. 203–226.

[38] Y. Gao, S. F. Al-Sarawi, and D. Abbott, "Physical unclonable functions," *Nature Electronics*, vol. 3, no. 2, pp. 81–91, 2020.

[39] I. Cisco Systems, "Cisco nexus 1000v layer 2 switching configuration guide," in 2013, ch. Configuring Private VLANs, Version 4.2(1)SV2(2.1). [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus1000/sw/4_2_1_s_v_2_2_1/layer2/b_Cisco_Nexus_1000V_Layer_2_Switching_Configuration_Guide_Release_4_2_1_SV_2_2_1/b_Cisco_Nexus_1000V_Layer_2_Switching_Configuration_Guide_Release_4_2_1_SV_2_2_1_chapter_0101.html.

[40] I. Cisco Systems, "Connected grid switch software configuration guide for ios," in 2013, ch. Configuring Asymmetric VLAN Mapping, Version 15.0(2)ED. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/connectedgrid/cgs2520/software/release/15_0_2_ed/configuration/guide/cgs_15_0_2ed/cgs_asym-vlan.html.

[41] Various, *Overview of the linux virtual file system*, Website. [Online]. Available: https://www.kernel.org/doc/html/latest/filesystems/vfs.html.

[42] L. W. Hollasch, A. Viviano, and M. Msft, *File system minifilter*, Website. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts.

[43] É. Ó. Muirí, "Framing software component transparency: Establishing a common software bill of material (sbom)," *NTIA, Nov*, vol. 12, 2019.

[44] S. Peisert, B. Schneier, H. Okhravi, *et al.*, "Perspectives on the solarwinds incident," *IEEE Security & Privacy*, vol. 19, no. 2, pp. 7–13, 2021.

[45] NTIA, *SBOM at a Glance*, https://tinyurl.com/txyvbhfu, Last Accessed: 15-10-2024.

[46] The United States Department of Commerce, *The Minimum Elements For a Software Bill of Materials (SBOM)*, https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf, Last Accessed: 15-10-2024.

[47] *Roles and Benefits for SBOM Across the Supply Chain*, https://www.ntia.gov/sites/default/files/publications/ntia_sbom_use_cases_roles_benefits-nov2019_0.pdf, Last Accessed: 15-10-2024.

[48] White House, *Executive Order on Improving the Nation's Cybersecurity*, https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/, Last Accessed: 05-03-2024.

[49] P. Hernandez, *Executive order 14028, improving the nation's cybersecurity*, 2021.

[50] Sonatype, *Biden executive order on cybersecurity calls for enhanced software supply chain security*, https://tinyurl.com/ywuyuzck, Last Accessed: 15-10-2024.

[51] European Parliament, *COMPROMISE AMENDMENT 1 - CYBER RESILIENCE ACT*, https://www.europarl.europa.eu/meetdocs/2014_2019/plmrep/COMMITTEES/ITRE/DV/2023/07-19/11-CA_CRA_EN.pdf, Last Accessed: 15-10-2024.

[52] *SPDX Overview*, https://spdx.dev/about/overview/, Last Accessed: 15-10-2024.

[53] *CycloneDx History*, https://cyclonedx.org/about/history/, Last Accessed: 15-10-2024.

[54] S. Feng and M. Lubis, "Defense-in-depth security strategy in log4j vulnerability analysis," in *2022 International Conference Advancement in Data Science, E-learning and Information Systems (ICADEIS)*, IEEE, 2022, pp. 01–04.

[55] J. E. Akinsola, A. S. Ogunbanwo, O. J. Okesola, I. J. Odun-Ayo, F. D. Ayegbusi, and A. A. Adebiyi, "Comparative analysis of software development life cycle models (sdlc)," in *Intelligent Algorithms in Software Engineering: Proceedings of the 9th Computer Science On-line Conference 2020, Volume 1 9*, Springer, 2020, pp. 310–322.

[56] A. Mishra and D. Dubey, "A comparative study of different software development life cycle models in different scenarios," *International Journal of Advance research in computer science and management studies*, vol. 1, no. 5, 2013.

[57] *SBOM's History*, `https://www.ntia.doc.gov/files/ntia/publications/uscc_-_2021.06.17.pdf`, Last Accessed: 15-10-2024.

[58] OWASP, *Dependency Track*, `https://dependencytrack.org/`, Last Accessed: 05-03-2024.

[59] DevOps Kung Fu, *Bomber*, `https://github.com/devops-kung-fu/bomber`, Last Accessed: 15-10-2024.

[60] Intel Corporation, *cve bin tool*, `https://github.com/intel/cve-bin-tool`, Last Accessed: 15-10-2024.

[61] *Grype*, `https://github.com/anchore/grype`, Last Accessed: 22-05-2024.

[62] *OWASP Dependency Track Vulnerability Data Source*, `https://owasp.org/www-project-dependency-track/`, Last Accessed: 15-10-2024.

[63] *Bomber Vulnerability Data Source*, `https://github.com/devops-kung-fu/bomber?tab=readme-ov-file#providers`, Last Accessed: 15-10-2024.

[64] *CveBinTool Vulnerability Data Source*, `https://github.com/intel/cve-bin-tool/blob/main/doc/MANUAL.md#data-sources`, Last Accessed: 15-10-2024.

[65] *Maven*, `https://maven.apache.org/`, Last Accessed: 15-10-2024.

[66] *Npm*, `https://docs.npmjs.com/about-npm`, Last Accessed: 15-10-2024.

[67] *The Python Package Index*, `hhttps://pypi.org/project/pip/`, Last Accessed: 15-10-2024.

[68] *Composer*, `https://getcomposer.org/`, Last Accessed: 15-10-2024.

[69] *Tiobe Index*, `https://www.tiobe.com/tiobe-index/`, Last Accessed: 15-10-2024.

[70] CycloneDx Python, *CycloneDx Python*, `https://github.com/CycloneDX/cyclonedx-python`.

[71] CycloneDx Conan, *CycloneDx Conan*, `https://github.com/CycloneDX/cyclonedx-conan`.

[72] CycloneDx Maven, *CycloneDx Maven*, `https://github.com/CycloneDX/cyclonedx-maven-plugin`, Last Accessed: 15-10-2024.

[73] CycloneDx Dotnet, *CycloneDx Dotnet*, `https://github.com/CycloneDX/cyclonedx-dotnet`, Last Accessed: 15-10-2024.

[74] CycloneDx Node Npm, *CycloneDx Node Npm*, `https://github.com/CycloneDX/cyclonedx-node-npm`, Last Accessed: 15-10-2024.

[75] CycloneDx PHP Composer, *CycloneDx PHP Composer*, `https://github.com/CycloneDX/cyclonedx-php-composer`, Last Accessed: 15-10-2024.

[76] *Syft*, `https://github.com/anchore/syft`, Last Accessed: 23-05-2024.

[77] CycloneDx Cona2n, *CycloneDx Conan2*, `https://github.com/conan-io/conan-extensions#readme`.

[78] CycloneDx Core (Java), *CycloneDx Core (Java)*, `https://github.com/CycloneDX/cyclonedx-core-java`, Last Accessed: 15-10-2024.

[79] CycloneDx Maven Configuration, *CycloneDx Maven Configuration*, `https://github.com/CycloneDX/cyclonedx-maven-plugin?tab=readme-ov-file#maven-usage`, Last Accessed: 15-10-2024.

[80] B. Laurie, "Certificate transparency," *Communications of the ACM*, vol. 57, no. 10, pp. 40–46, 2014.

[81] B. Laurie, "Certificate transparency: Public, verifiable, append-only logs," *Queue*, vol. 12, no. 8, pp. 10–19, 2014.

[82] N. Van der Meulen, "Diginotar: Dissecting the first dutch digital disaster," *Journal of strategic security*, vol. 6, no. 2, pp. 46–58, 2013.

[83] S. Haber and W. S. Stornetta, *How to time-stamp a digital document*. Springer, 1991.

[84] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Satoshi Nakamoto*, 2008.

[85] Z. Wang, J. Lin, Q. Cai, Q. Wang, D. Zha, and J. Jing, "Blockchain-based certificate transparency and revocation transparency," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 681–697, 2020.

[86] D. Madala, M. P. Jhanwar, and A. Chattopadhyay, "Certificate transparency using blockchain," in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, IEEE, 2018, pp. 71–80.

[87] G. Korkmaz, J. B. S. Calderón, B. L. Kramer, L. Guci, and C. A. Robbins, "From github to gdp: A framework for measuring open source software innovation," *Research Policy*, vol. 53, no. 3, p. 104 954, 2024.

[88] *Blockchain Size (MB)*, `https://www.blockchain.com/explorer/charts/blocks-size/`, Last Accessed: 21-06-2024.

[89] *This year in JavaScript: 2018 in review and npm's predictions for 2019*, `https://blog.npmjs.org/post/180868064080/this-year-in-javascript-2018-in-review-and-npms/`, Last Accessed: 24-06-2024.

[90] *Average Confirmation Time*, `https://www.blockchain.com/explorer/charts/avg-confirmation-time/`, Last Accessed: 23-06-2024.

[91] B. Xia, D. Zhang, Y. Liu, Q. Lu, Z. Xing, and L. Zhu, "Trust in software supply chains: Blockchain-enabled sbom and the aibom future," in *Proceedings of the 2024 ACM/IEEE 4th International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS) and 2024 IEEE/ACM Second International Workshop on Software Vulnerability*, 2024, pp. 12–19.

[92] *Deliver Uncompromised: Securing Critical Software Supply Chains*, `https://tinyurl.com/ydxzcw78`, Last Accessed: 15-10-2024.

[93] L. J. Camp and V. Andalibi, "Sbom vulnerability assessment & corresponding requirements," *NTIA Response to Notice and Request for Comments on Software Bill of Materials Elements and Considerations*, 2021.

[94] *Supply Chain Compromise: Compromise Software Supply Chain*, `https://attack.mitre.org/techniques/T1195/002/`, Last Accessed: 15-10-2024.

[95] J. Yang, Y. Lee, and A. P. McDonald, "Solarwinds software supply chain security: Better protection with enforced policies and technologies," *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing 22*, pp. 43–58, 2022.

[96] G. Lu, X. Koufteros, and L. Lucianetti, "Supply chain security: A classification of practices and an empirical study of differential effects and complementarity," *IEEE Transactions on Engineering Management*, vol. 64, no. 2, pp. 234–248, 2017.

[97] N. F. Syed, S. W. Shah, R. Trujillo-Rasua, and R. Doss, "Traceability in supply chains: A cyber security analysis," *Computers & Security*, vol. 112, p. 102 536, 2022.

[98] S. E. Chang and Y. Chen, "When blockchain meets supply chain: A systematic literature review on current development and potential applications," *Ieee Access*, vol. 8, pp. 62 478–62 494, 2020.

[99] P. Dutta, T.-M. Choi, S. Somani, and R. Butala, "Blockchain technology in supply chain operations: Applications, challenges and research opportunities," *Transportation research part e: Logistics and transportation review*, vol. 142, p. 102 067, 2020.

[100] S. Al-Farsi, M. M. Rathore, and S. Bakiras, "Security of blockchain-based supply chain management systems: Challenges and opportunities," *Applied Sciences*, vol. 11, no. 12, p. 5585, 2021.

[101] M. N. Halgamuge, "Leveraging deep learning to strengthen the cyber-resilience of renewable energy supply chains: A survey," *IEEE Communications Surveys & Tutorials*, 2024.

[102] E. N. Akimova, A. Y. Bersenev, A. A. Deikov, *et al.*, "A survey on software defect prediction using deep learning," *Mathematics*, vol. 9, no. 11, p. 1180, 2021.

[103] S. Stradowski and L. Madeyski, "Industrial applications of software defect prediction using machine learning: A business-driven systematic literature review," *Information and Software Technology*, vol. 159, p. 107 192, 2023.

[104] P. Radanliev, "In software vulnerability management: Automations in the software bill of materials (sbom) and the vulnerability-exploitability exchange (vex)," *Authorea Preprints*, 2023.

[105] H. Kim and K.-H. Kim, "Deep learning-based sbom defect detection for medical devices," in *2024 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, IEEE, 2024, pp. 047–051.

This project is supported by the Belgian Energy Transition Funds