University of St. Gallen

School of Computer Science

to obtain the title of

Bachelor of Science in Computer Science

Bachelor Thesis on

AI-Driven Secure IoT Onboarding via MUD based Segmentation in Industrial Networks

Building an automated pipeline for MUD file assessment

Submitted by:

Vito Emanuel Steiner

22-609-424

Approved on the application by:

Prof. Dr. Bruno Rodrigues

Date of Submission:

Mai 19, 2026

Abstract

The Internet of Things (IoT) introduces significant security risks to local networks. The Manufacturer Usage Description (MUD) standard mitigates these risks by restricting devices to their intended network traffic using Zero Trust whitelists specified by the device manufacturer. However, blindly onboarding MUD files is dangerous. It risks integrating flawed or malicious configurations directly into the local network.

This thesis introduces an automated, five-stage pipeline based on related work in the field to analyze MUD files before deployment for their technical readability and file accuracy (correctness and comprehensiveness). The architecture combines deterministic checks, YANG schema validation, and systematic conformance testing, with a vulnerability library ruleset and an AI-driven expert assessment using a Large Language Model (LLM). The goal is not only to evaluate the effectiveness of such a pipeline but also the capabilities and limitations of an LLM as a judge for semantic plausibility.

Evaluation shows a clear duality in effectiveness. Deterministic stages are highly reliable for syntax and conformance checks, while the LLM (Gemini 2.5 Pro) is strong at detecting over-specification and known high-risk patterns. However, it is unreliable to detect under-specified MUD profiles. Moreover, it is vulnerable to semantic deception through benign rule naming, which is a new attack surface introduced when replacing human judgement with an LLM.

Consequently, fully autonomous AI-based onboarding is not yet robust enough for secure deployment. The most viable path is a hybrid architecture: deterministic safeguards as the safety baseline, augmented by LLM-based contextual reasoning as an analytical assistant. Future work should benchmark human and machine judgement directly and combine static pre-deployment analysis based on improved rulesets with dynamic runtime monitoring to detect omissions that static checks miss.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The growth of the Internet of Things and its expanding threat surface

1.1.1 The history and growth of the Internet of Things

Since the term “Internet of Things” (IoT) was coined in 1999, the domain has seen rapid growth, increasing from under USD 2 billion in 2018 to nearly USD 1 trillion by 2023, and reaching a multi-trillion-dollar scale by the mid-2020s. This expansion is driven by continuous advances in connectivity, cloud computing, and embedded systems, enabling a steady stream of new smart applications. Today, IoT devices outnumber the global human population and are widely deployed across domains such as home automation, smart cities, healthcare, industrial manufacturing, and agriculture, highlighting their growing role in both consumer and industrial ecosystems [1].

1.1.2 Characteristics and vulnerabilities of IoT devices

Unlike general-purpose computers, IoT devices are task-specific, resource-constrained units. They operate on minimal processing power and simplified operating systems incapable of supporting resource-intensive security protocols. Security is frequently secondary to cost-efficiency in mass-produced devices. This creates a “security-by-design” deficit where protective measures are sacrificed for low price points and rapid time-to-market [2].

The inherent characteristics of IoT ecosystems create five primary security challenges [3]:

1. Low-cost production: No-brand manufacturers prioritize cost over security design.

2. Hardware constraints: Limited memory prevents standard cryptographic operations. Unoptimized open-source software leaves unnecessary ports active.

3. Heterogeneity: Lack of standardized hardware and interfaces makes universal security protocols technically unfeasible.

4. Unattended physical deployment: Remote IoT devices are vulnerable to physical tampering or side-channel attacks.

5. Administrative negligence: The "plug-and-play" nature encourages poor security hygiene. Users rarely change default credentials, and devices lack automated patching mechanisms.

The 2016 Mirai Botnet is the definitive example of exploited IoT vulnerabilities. Mirai scanned the public IPv4 space for open Telnet ports. It compromised hundreds of thousands of IP cameras and routers using a simple dictionary of 62 default username/password pairs and turned these into a massive botnet of IoT devices [4].

The attack demonstrated two critical risks: poor configuration hygiene and unrestricted connectivity. Once compromised, these devices established arbitrary outbound connections to a Command-and-Control infrastructure. Because the devices' communication was not restricted to their intended behavior, they were able to launch massive Distributed Denial-of-Service (DDoS) attacks, culminating in the outage of the DNS provider "Dyn" [4]. MUD would have directly neutralized this vulnerability. By strictly whitelisting only intended destinations, a MUD-enforced network would have prevented the botnet from establishing outbound connections to Command-and-Control servers, and thereby blocked the botnet's attack vector.

1.2 Operational challenges in network security

1.2.1 The failure of manual access control and garden walls

Historically, network security relied on administrators manually defining Access Control Lists (ACLs) to establish perimeter defenses, often referred to as "garden walls". While functional for traditional IT endpoints, the sheer volume and complexity of IoT devices render manual management impossible. Each device requires a highly specific matrix of connections to cloud services, update servers, and local peers [5].

1.2.1.1 Resource constraints for Small and Medium Businesses

Moreover, Small and Medium Businesses (SMBs) often lack the financial capital, specialized staff, and technical infrastructure required to maintain a robust cybersecurity posture. Security is frequently perceived as secondary to core operations[6]. This is relevant for Switzerland, where SMBs make up 99% of all companies [7].

1.2.2 The paradigm shift

The failure of the traditional perimeter-based approach led to the emergence of Zero Trust Architecture (ZTA). ZTA operates on the principle that no user or device is trusted by default, regardless of its network location [8].

Zero Trust stops unauthorized access by enforcing granular, "least privilege" access decisions on a per-request basis in a network [9]. This shifts the focus from protecting network zones to micro-segmenting individual resources.

1.2.3 Manufacturer-declared device behavior via MUD

The Manufacturer Usage Description (MUD) standard directly transfers the Zero Trust paradigm to the IoT domain [10].

It shifts the responsibility for defining network access policies from the local administrator to the device manufacturer. Instead of manual configuration, MUD utilizes a machine-readable, whitelist-based MUD file provided by the manufacturer. This file specifies the exact network communications required for a device's intended function. When an IoT device connects, the network automatically downloads this file and translates the manufacturer's intent into strict, Zero Trust access policies. This approach efficiently micro-segments the device without manual administrative overhead. [11]

1.3 Problem statement

The MUD standard resolves operational scalability but introduces a new risk: network security becomes entirely dependent on external policy files. The standard relies on digital signatures to verify a file's origin and integrity, but it fails to evaluate the actual security implications of the policies [10]. We operate under the assumption that this cryptographic mechanism is insufficient, as it inherently excludes unsigned files from low-cost manufacturers or covers misspecified files by accident. Relying solely on signatures leaves a critical gap in network defense.

There is currently no standardized mechanism to check if a manufacturer, whether through negligence or malicious intent, is introducing potentially harmful policies. Crucially, malicious actors might intentionally distribute over-permissive files to bypass superficial inspections. By automatically applying these files, local networks blindly cede control of their security perimeter to external third parties. This trust delegation transforms the MUD file itself into a possible attack vector. A compromised manufacturer could weaponize a miss-specified profile to grant attackers unbound movement across an internal network.

Consequently, "blind onboarding" creates a critical vulnerability. To ensure secure IoT integration systems require a pre-deployment analysis framework. This thesis addresses this problem by investigating how software solutions can automatically analyze MUD files prior to onboarding. Specifically, it assesses the viability and effectiveness of using an Artificial Intelligence (AI) component to evaluate the semantic security of intended policies in these manufacturer-provided files.

1.4 Research questions

To evaluate the automated security analysis of MUD files, this thesis addresses the following three research questions:

1. How can MUD files be analyzed for security risks before deployment?
2. How could such an analysis be automated with software?
3. How effectively can the resulting software analyze security in MUD files?

1.5 Contributions

This thesis contributes to the field of IoT network security in four distinct areas:

1. The conceptual design of an analytical pipeline capable of parsing, checking, and evaluating MUD files.
2. A proof-of-concept software implementation that serves to automate the aforementioned pipeline.
3. The refinement of the systematic MUD conformance framework established by [12], which involved the identification and correction of two misspecified test procedures to ensure strict alignment with the MUD standard.

4. A final empirical evaluation of the pipeline's effectiveness, specifically addressing the capabilities and limitations of the AI-driven analysis component within the overall architecture.

1.6 Thesis structure

The remainder of this thesis is structured as follows. Chapter 2 "Background" introduces the foundational concepts of the MUD standard and Large Language Models. Chapter 3 "Related Work" reviews existing literature on MUD file validation and the use of AI for security analysis. Chapter 4 "Methodology" explicitly outlines the framework and the design of the evaluation pipeline. Chapter 5 "Implementation" details the technical development of the proof-of-concept application. Chapter 6 "Evaluation" systematically assesses the pipeline's performance. Finally, Chapter 7 "Discussion" discusses the implications of the findings, and Chapter 8 "Future Work" outlines potential avenues for future research. The thesis concludes with a summary of contributions and final thoughts in Chapter 9 "Conclusion".

Chapter 2

Background

2.1 Zero trust architecture

Zero Trust Architecture (ZTA) operates on the principle that no user or device is trusted by default, regardless of its network location. According to NIST [9], Zero Trust is not a specific product but a collection of guiding principles for securing an organization's infrastructure. NIST provides the following operative definition:

> Zero trust (ZT) provides a collection of concepts and ideas designed to minimize uncertainty in enforcing accurate, least privilege per-request access decisions in information systems and services in the face of a network viewed as compromised. Zero trust architecture (ZTA) is an enterprise's cybersecurity plan that utilizes zero trust concepts and encompasses component relationships, workflow planning, and access policies [9].

This definition emphasizes granular, "least privilege" access control. According to [8], Zero Trust relies on four key principles:

1. Authentication and access control: Every request to access a resource must be verified.
2. Lightweight encryption: All communications must be protected.
3. Micro-segmentation and software-defined perimeters: The network is divided into isolated pieces to prevent lateral attacker movement.
4. Dynamic and context-aware security orchestration: Security policies automatically adjust based on current conditions.

Because IoT devices are typically task-specific and resource-constrained, they cannot run their own Zero Trust enforcement. ZTA cannot be directly executed on the device itself. Instead, the surrounding network infrastructure must enforce micro-segmentation and least-privilege access on the device's behalf.

2.2 The Manufacturer Usage Description standard

This section provides an overview of the Manufacturer Usage Description (MUD) specification. Unless otherwise noted, the technical details, architecture, and terminology described herein are based on the official specification and documentation [11].

2.2.1 Overview and architectural components

The Manufacturer Usage Description (MUD) standard, codified as RFC 8520 by the Internet Engineering Task Force (IETF), allows manufacturers to formally state an IoT device's required communication patterns. By explicitly defining intended behavior, the network automatically blocks all other traffic. This approach scales security across massive device deployments and protects legacy systems that no longer receive firmware updates.

The MUD framework relies on three core components:

1. a `URL` emitted by the device, acting as a pointer to the manufacturer's server,
2. a machine-readable `MUD file` that contains the policy definition, and
3. a `MUD manager` that retrieves the MUD file and translates it into enforceable network policies.

2.2.2 MUD processing workflow

The MUD process begins when a device announces its MUD URL (e.g., via a DHCP option, an X.509 certificate extension, or an LLDP frame).

fig. 2.1 displays the basic information flow within the MUD framework. The sequence is as follows:

1. The IoT device announces its MUD URL.
2. The router forwards the MUD URL to the MUD manager.
3. The MUD manager retrieves the MUD file and its cryptographic signature from the manufacturer's server.

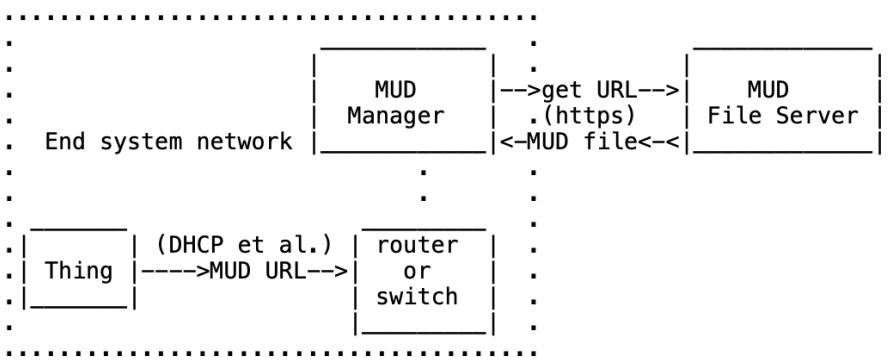

Figure 2.1: MUD processing workflow, copied from [11]

Upon successful signature validation, the MUD manager executes the following tasks:

1. Translates the MUD file abstractions into specific network configurations.
2. Updates hardware elements (e.g., switches, firewalls) with the appropriate access control configurations.

How the MUD manager exactly conducts these steps is up to the implementation. A MUD Manager can be a dedicated service or a pre-implemented component of an AAA system or a network management system.

2.2.3 MUD YANG model and semantic abstractions

A MUD file is a structured JSON document governed by a YANG model. This model defines the data hierarchy and constraints. It ensures every MUD file follows a predictable, machine-readable format for network management systems to parse. The MUD YANG model consists of three components:

1. `The MUD container`: Used for metadata and policy pointers.
2. `ACL match augmentations`: Provides abstractions like “manufacturer” or “controller” instead of static IP addresses.
3. `TCP direction augmentations`: Facilitates stateful security configuration.

The JSON document contains two primary root objects.

- The `ietf-mud:mud container` acts as the document header. It contains mandatory metadata such as the version, the unique MUD URL, timestamps, and pointers to ACL definitions.

- The `ietf-access-control-list:acls container` acts as the document body. It defines the access control policies. This section specifies Access Control Lists (ACLs). Each ACL contains Access Control Entries (ACEs) that define the permitted device rules.

Manufacturers cannot predict specific IP addresses or local configurations of an end-user network. Consequently, MUD files use abstractions as placeholders within firewall rules. These are based on the device's unique MUD URL:

$$\texttt{https://}\underbrace{\texttt{vendor-for-iot.com}}_{\text{Authority}}\texttt{/}\underbrace{\texttt{mud-files/devicea-model2}}_{\text{Path}}$$

- `model` (Exact MUD URL Match)
  The most restrictive scope. It permits communication only with devices possessing the exact same MUD URL. If Device A and Device B share the same URL, they are recognized as identical models.

- `same-manufacturer` (Authority Match)
  Targets the *Authority* component. It allows communication with any other device from the same vendor (e.g., any device starting with `vendor-for-iot.com`), regardless of the specific model.

- `manufacturer` (Third Party Vendor Match)
  Allows communication with a specific *other* manufacturer. The MUD Manager identifies all devices on the network belonging to that specified authority (e.g., `"partner-vendor.com"`).

- `controller` (URI-Based Service)
  Permits communication with a specific class of network service (e.g., `uri:local:safety-controllers`). The MUD Manager resolves this URI to the local IP of that service.

- `my-controller` (Boolean Flag)
  A YANG `empty` type flag indicating the device must talk to its primary designated controller. The MUD Manager populates this based on local network pre-configuration.

- `local-networks` (Boolean Flag)
  A YANG `empty` type flag that restricts the device to the local segment, effectively "air-gapping" it from the public internet.

```
module: ietf-mud
  +--rw mud!
     +--rw mud-version           uint8
     +--rw mud-url               inet:uri
     +--rw last-update           yang:date-and-time
     +--rw mud-signature?        inet:uri
     +--rw cache-validity?       uint8
     +--rw is-supported          boolean
     +--rw systeminfo?           string
     +--rw mfg-name?             string
     +--rw model-name?           string
     +--rw firmware-rev?         string
     +--rw software-rev?         string
     +--rw documentation?        inet:uri
     +--rw extensions*           string
     +--rw from-device-policy
     |  +--rw acls
     |     +--rw access-list* [name]
     |        +--rw name    -> /acl:acls/acl/name
     +--rw to-device-policy
        +--rw acls
           +--rw access-list* [name]
              +--rw name    -> /acl:acls/acl/name

  augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches:
    +--rw mud
       +--rw manufacturer?        inet:host
       +--rw same-manufacturer?   empty
       +--rw model?               inet:uri
       +--rw local-networks?      empty
       +--rw controller?          inet:uri
       +--rw my-controller?       empty
  augment
    /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches
       /acl:l4/acl:tcp/acl:tcp:
    +--rw direction-initiated?   direction
```

Figure 2.2: YANG tree of the MUD model, copied from [11]

It is important to note that `controller` and `my-controller` are placeholders. The MUD Manager is responsible for populating these rules by resolving the URIs or flags into the actual, current IP addresses used within the local network infrastructure.

The complete YANG tree of the mud model can be found in fig. 2.2.

The in-detail documentation of the YANG-model, including its structural components, leaf nodes, and associated descriptions, can be found in the official RFC 8520 web-editor [1] for quick reference.

[1] https://www.rfc-editor.org/rfc/rfc8520.txt

2.2.4 MUD signatures and validation

To ensure policy integrity, the MUD standard requires cryptographic signature validation. The MUD manager retrieves a separate signature file identified by the `mud-signature` field. It verifies this signature against the MUD document using a digital certificate with an active digital signature bit. The signature must match the authorized signer. If validation fails, the manager halts the process and awaits manual administrator approval.

The primary goal of this signature is accountability. It allows administrators to verify the reputation of the entity recommending the security rules. The standard acknowledges that trusted entities like system integrators can also sign files. This focuses on the reliability of the security recommendation rather than the device brand.

Administrators use standard cryptographic tools, such as OpenSSL, to verify the file integrity and source trust. [11] provides an example for OpenSSL signature verification:

Listing 2.1: Example OpenSSL signature verification, copied from [11]

```
2.1.1 openssl cms -verify -in mudfile.p7s -inform DER -content mudfile
```

2.2.5 MUD security considerations

The MUD standard addresses the risk of devices providing false network requirements to bypass security perimeters.

To counter faulty or maliciously crafted MUD files designed to exploit vulnerabilities, the RFC recommends two defense strategies:

- Validating the signer against a pre-established trusted system hierarchy.
- Executing a preliminary analytical scan to ensure the file is technically parsable and behaviorally “believable”.

We judge the first defense strategy to be inadequate on its own. A valid signature guarantees origin, but compromised or negligent manufacturers can still sign maliciously crafted policies.

The automated scan verifying technical parsability and behavioral “believability” forms the central focus of this thesis.

2.3 Artificial Intelligence

2.3.1 Definition of Artificial Intelligence

If we had to define Artificial Intelligence (AI) in simple words, we could put it as machines which are able to think and learn like humans. A comprehensive legal and technical definition is provided by the EU AI Act, which defines AI as:

> a machine-based system that is designed to operate with varying levels of autonomy and that may exhibit adaptiveness after deployment, and that, for explicit or implicit objectives, infers, from the input it receives, how to generate outputs such as predictions, content, recommendations, or decisions that can influence physical or virtual environments [13].

2.3.2 Generative AI and Large Language Models

The current technical frontier is Generative AI, a specific form of deep learning capable of learning input patterns to synthesize novel output. A dominant subset of Generative AI is Large Language Models (LLMs), deep neural architectures specialized for generating human language. LLMs predict the most likely next token based on statistical patterns learned from massive datasets.

As foundation models, LLMs excel at executing complex, language-based reasoning tasks. However, their primary risk is “hallucination”, which is the generation of plausible but factually incorrect outputs. Because LLMs lack an active fact-checking mechanism, their knowledge is mathematical and statistical, derived from language learning rather than derived from objective fact comprehension [14]. This statistical guesswork could be fatal in security engineering. A hallucinating model might confidently validate a malicious access, make up a legitiate explanation for a miss-specified rule or invent a phantom threat, thereby silently compromising the entire Zero Trust perimeter.

2.3.3 Taxonomy of LLMs, AI Agents, and Agentic AI

To operationalize LLMs beyond text generation, the field introduced autonomous system architectures. It is necessary to distinguish between reactive LLMs, isolated AI Agents, and Agentic AI systems.

The primary characteristic of a standard LLM is reactivity. Operations are triggered exclusively by user prompts. They are stateless, lack goal-following mechanisms, and cannot independently act upon their environment.

By contrast, an AI Agent is an autonomous software entity built around an LLM reasoning engine. It operates in a loop: perceiving inputs, reasoning through a plan, utilizing external tools (e.g., executing code, fetching web data), and learning from the results. AI Agents are optimized for specific, goal-directed tasks and operate in isolated autonomy.

Agentic AI represents a paradigm shift to coordinated, multi-agent frameworks. Instead of one agent solving one task, an orchestrator decomposes high-level goals and assigns sub-tasks to specialized agents. These systems keep longer context and support long-horizon planning [15].

Table 2.1: Comparison of LLMs, AI Agents, and Agentic AI, adapted from [15]

Dimension	LLM	AI Agent	Agentic AI
Initiation	**Reactive:** Triggered by a specific prompt	**Goal-triggered:** Uses perception, reasoning, and tools to meet explicit instructions	**Orchestrated:** Proactively decomposes and manages workflows
Autonomy	**Low**	**Medium**	**High**
Memory	**Stateless**	**Short-term**	**Long-term**
Planning scope	**None:** Next-token prediction	**Single-step:** Narrow, well-defined task execution	**Multi-step:** Decomposes large goals into distributed sub-tasks
Interaction	User → LLM → Output	User → Agent → Tool → Output	User → Orchestrator → Multiple Agents → Output
Key characteristic	**Reactivity**	**Tool use**	**Collaboration**

2.3.4 Choosing the right AI system level

Choosing the right system level should be a design decision, not only an implementation detail. The chosen level defines capability boundaries, operational risk, and engineering cost. If the level is too low, critical dependencies remain unmanaged and task quality degrades. If the level is too high, systems become harder to validate, operate, and secure. Consequently, selecting the minimum level according to Table reftab:system-recommendations that satisfies task requirements reduces both under-engineering and over-engineering.

Table 2.2: system level Recommendations based on Required Capabilities, adapted from [15]

Required Capability	**Recommended system level**	**Reason**
Content Creation / Task specific reasoning	reactive LLM	Best for reactive tasks, triggered by a single input
Tool based Execution	AI Agent	Best for narrow, domain-specific tasks which require environment perception and tool calling
Workflow orchestration	Agentic AI System	Best for complex, high-level goals that must be decomposed into sub-tasks and managed across specialized roles.

Chapter 3

Related work

3.1 The MUD ecosystem and literature landscape

Existing academic MUD research can be grouped into four areas: profile generation, file obtaining, file enforcement, and MUD-based applications [16]. Existing end-to-end MUD deployments already onboard devices at scale, for example in Cisco and NIST implementations. However, these workflows mainly verify signature authenticity and transport integrity. They rarely validate whether policy content is semantically plausible for the device function [10].

3.2 The MUD lifecycle and the analysis gap

The standard MUD file life-cycle is modeled in three stages: generation, obtaining, and enforcement.

1. Generation: The manufacturer defines intended communication behavior, either manually or from observed traffic traces.

2. Obtaining: When a device joins a network, it advertises a MUD URL (for example via DHCP, LLDP, or X.509), and the MUD manager retrieves the file.

3. Enforcement: The MUD manager translates abstract rules into concrete network controls [16].

This model leaves a structural gap between obtaining and enforcement: an explicit validation phase.

Furthermore, profiles derived from traffic observations can encode malicious behavior as legitimate behavior if the device was already compromised during the observation and generation process. In that case, enforcement institutionalizes overprivileged or harmful

communication. Consequently, pre-deployment plausibility analysis is required before policy activation [17].

3.3 Predeployment analysis of MUD files

To bridge the gap, the literature has brought forward various tools to analyze MUD files before enforcement. These predeployment validation frameworks operate across four distinct approaches:

- Structural and Mathematical Validation: Frameworks like MUDdy [18] use formal logic to prove internal consistency with mathematical encoded security policies of the target environment.
- Comparative Assessment: Tools such as MUDIS [19] and automated pre-certification [20] assess profile believability by comparing intended rules against historical baselines or live network traffic.
- Standard Conformance: The framework of [12] systematically audits MUD files against the RFC 8520 standard, quantifying compliance through a comprehensive test suite.
- Human-Augmented Analysis: Systems like MUD-Visualizer [21] abstract complex JSON syntax into visual flow graphs, explicitly leveraging human expertise to determine the functional plausibility of a profile.

The following sections examine prominent implementations across these dimensions, highlighting both their capabilities and their inherent limitations in fully automating semantic validation.

3.3.1 Formal semantic verification: MUDdy

To address the accuracy gap mathematically, [18] introduce MUDdy, a framework for formal semantic verification. MUDdy models access-control policies as metagraphs to validate internal consistency and organizational compatibility before a profile is enforced.

The tool executes a sequential pipeline. First, it verifies structural compliance with RFC 8520. Second, it utilizes metagraph algebra to identify redundant or overlapping policy rules within the profile. Third, it evaluates profile safety through mathematical

inclusion. By comparing vectorized policies, MUDdy verifies whether the manufacturer-defined device behavior forms a strict subset of the target network's allowed security policy [18].

While this approach formally guarantees compliance against a specific environment, it requires administrators to explicitly model and formalize their internal network zones. This prerequisite creates a high operational burden, limiting its feasibility for organizations lacking dedicated formalization expertise.

3.3.2 Cross-contextual validation: MUDIS

Shifting the focus from internal consistency to external behavior, the MUD Inspection System (MUDIS) approaches the accuracy gap through cross-contextual validation. [19] evaluate profile believability by comparing a device's current MUD file against historical baselines, such as previous firmware versions or different network deployments.

To quantify anomalies, MUDIS categorizes Access Control Entries (ACEs) based on structural variance and calculates a Similarity Score using the Jaccard coefficient. This metric allows administrators to detect whether a new profile deviates suspiciously from a known "gold standard". Consequently, while MUDdy relies on strict mathematical set inclusion against formal network zones, MUDIS provides a granular similarity comparison to identify anomalous behavioral shifts over a device's life-cycle [19].

3.3.3 Systematic verification of conformance to the MUD standard

Rather than evaluating profiles relative to organizational policies or environmental contexts, [12] assess the MUD Device and File in strict isolation. They propose a methodology to quantify adherence to the RFC 8520 standard, creating a predictable baseline for protocol conformance.

The core of this approach maps mandatory and optional RFC directives into a 26-test suite. These tests verify the entire signaling chain: the device's ability to emit a valid MUD URL, the infrastructure's capacity to securely host the signature, and the file's structural and logical correctness.

To accommodate varying manufacturer capabilities, the framework introduces a staged adoption hierarchy consisting of four main tiers. The temporary "Bronze" rank identifies devices in a preparatory phase that merely emit a MUD URL, irrespective of its validity. For permanent, usable profiles, the "Silver" tier establishes the baseline by demanding a structurally and logically complete file. The "Gold" tier strictly requires

a verifiable digital signature and finally, the "Platinum" rank is awarded to profiles that successfully implement advanced optional features, such as IETF-standardized extensions. This isolation-based testing provides regulators and network operators with a quantifiable metric to mandate MUD adoption, systematically minimizing the attack surface before a device accesses the network [12].

3.3.4 Automation of IoT pre-certification security testing environment based on MUD

Moving beyond theoretical mathematical modeling, [20] present an adaptive pre-certification framework that compares intended MUD profiles against actual observed network traffic. Validating manual testing as highly laborious and unscalable, they propose an automated "quality check" prior to device deployment.

The architecture translates JSON-based MUD files and PCAP-based traffic logs into a unified "Service Description" format. It then executes a dual-track assessment: strict compliance testing to identify deviations between claimed and observed behaviors, and a vulnerability assessment against a standardized library (V-Library) derived from OWASP and NIST, which explicitly flags the occurrence of known vulnerabilities.

Crucially, [20] concede that while syntax validation and vulnerability checks can be fully automated, determining the strict functional plausibility of an allowed service (e.g., identifying if a printer legitimately requires BGP routing) remains challenging. Since the framework relies on generalized tests rather than device-specific semantic profiles, resolving overprivileged but technically valid rules still requires expert human insight.

3.3.5 Human-augmented analysis: MUD-Visualizer

This is the specific challenge MUD-Visualizer, a tool proposed by [21], seeks to resolve. Operating on the premise that exhaustive manual JSON analysis is beyond human cognition, the tool translates MUD files into interactive, visual flow graphs. By doing so, MUD-Visualizer tries to enable better human-in-the-loop decision-making to efficiently spot overprivileged behavior.

To prevent visual clutter, the tool first optimizes the raw profile. It merges interacting Access Control Entries (ACEs) and prunes redundant rules. The distilled logic is then rendered as a node-based graph, allowing administrators to visually pinpoint anomalous cross-device communications.

Empirical user studies validate this approach. [22] demonstrate that visual analysis doubles anomaly detection accuracy and halves analysis time compared to manual text

review. By abstracting the syntax layer, MUD-Visualizer proves that human intuition remains highly effective for determining functional plausibility when data is presented correctly.

3.4 The semantic plausibility bottleneck

The synthesis of existing research demonstrates that the structural and comparative dimensions of MUD file validation are well-established. Systematic auditing tools like [12] successfully automate syntax and RFC conformance, while vulnerability assessments like [20] reliably flag explicit vulnerabilities and deprecated encryption standards.

However, a critical bottleneck remains in automating semantic plausibility. Evaluating whether a requested service is logically appropriate for a specific device (e.g., "Should a basic printer utilize BGP routing?") cannot be solved by deterministic mathematical logic alone. Current frameworks either circumvent this issue by demanding extensive manual network modeling [18] or explicitly defer the decision to human experts by visualizing the data [22].

Reflecting on these findings, it is important to note that the state of the literature remains fundamentally fragmented. While the isolated tools discussed above effectively answer Research Question 1 by proving that pre-deployment analysis is technically possible, they fail to deliver a unified, functional operational workflow. For instance, while MUDdy [18] provides excellent mathematical certainty, it forces administrators into an impossible trade-off by requiring hours of manual labor to model internal network zones. Conversely, approaches like MUD-Visualizer [21] successfully make the data comprehensible but still ultimately rely on a human-in-the-loop to make the final judgment. Interestingly, this fragmentation of tools for safe deployment undermine the very scalability that the MUD standard promises to deliver in the first place.

Consequently, the domain remains open for fully automated deployments. To bridge this gap, we shall try to architect a unified pipeline. A solution should combine the available analysis approaches into a single workflow that scales without constant human intervention.

3.5 LLMs for security analysis and judgement

A new research frontier positions Large Language Models (LLMs) as a scalable substitute for human security experts. Unlike deterministic rule-based software, state-of-the-art LLMs can execute multi-step reasoning to interpret how low-level implementation details

impact high-level security postures [23]. As established in Chapter 1 “Introduction”, the exponential growth of IoT devices renders manual network administration impossible. To achieve true operational scalability, we must replace human intuition with machine cognition. As outlined in Chapter 2 “Background”, LLMs are uniquely equipped for this capability. They excel at human like reasoning capabilities, which makes them suitable for effectively automating the human-in-the-loop. But before taking this step, we shall first explore the capabilities of LLMs for security analysis and judgement.

3.5.1 LLM-driven configuration validation

Recent literature provides strong evidence that LLMs can effectively interpret, generate, and validate complex security configurations. [24] demonstrated this generative capability by utilizing LLMs to derive MUD profiles directly from C source code. By analyzing the usage contexts of socket APIs and port constants, their approach allows to model a structurally valid MUD profile. Crucially, generating profiles from source code avoids the risk of traffic-based profiling a compromised device.

Beyond generation, LLMs have proven capable of translating and validating semantic intent. For instance, the LLM-NetCFG framework translates high-level natural language intents into precise, executable device configurations, such as Access Control Lists (ACLs) or OSPF routing rules for multi-vendor hardware [25].

Directly addressing the configuration validation gap is the Ciri framework [26], which leverages LLMs to detect software misconfigurations that bypass standard syntax checks. Ciri actively audits semantic requirements and reasons through complex relationships between dependent variables to identify “silent” misconfigurations. This confirms that LLMs can successfully execute the qualitative, cross-contextual judgement that deterministic analysis tools currently lack.

3.5.2 Empirical benchmarking of LLM judgement

The viability of substituting human expertise with LLM evaluation is supported by rigorous empirical benchmarking. The SECURE benchmark, designed specifically for safety-critical domains such as Industrial Control Systems (ICS), evaluates LLMs acting autonomously as cybersecurity advisors. The results demonstrate that state-of-the-art models, particularly GPT-4, consistently outperform human subjects, including industry professionals and doctoral researchers, in knowledge-intensive security tasks [27].

Furthermore, the broader paradigm of utilizing an "LLM-as-a-judge" has proven highly robust for open-ended, qualitative assessments. In complex software engineering contexts, LLM-based evaluators achieve Pearson correlations exceeding 80% with human ground-truth scores. This capability significantly surpasses conventional, deterministic evaluation metrics, confirming that LLMs can reliably replicate human qualitative reasoning and preference alignment at scale [23].

3.5.3 Architectural challenges and mitigation strategies

Despite these robust capabilities, deploying LLMs for automated security judgements introduces specific architectural challenges. Models frequently exhibit agreement bias and may confidently hallucinate responses when encountering novel threats absent from their training data. While enforcing explicit, multi-step logic via Chain-of-Thought (CoT) prompting severely reduces these hallucinations, it can escalate computational overhead by up to 35 times [27].

To counteract these domain-specific limitations, researchers utilize targeted prompt engineering architectures. In risk assessment contexts, Retrieval-Augmented Generation (RAG) pipelines dynamically anchor the LLM to formal security standards, ensuring outputs remain strictly compliant with industry best practices rather than probabilistic guesses [28]. Additionally, evaluating configuration validation frameworks like Ciri confirms that few-shot prompting (supplying the model with explicit examples of both valid and malicious misconfigurations) achieves the highest detection accuracy [26].

3.5.4 Synthesis and research gap

Current deterministic tools for MUD analysis excel at syntactic and protocol conformance but fail to autonomously assess semantic plausibility, necessitating manual intervention. In parallel, Large Language Models have demonstrated high efficacy in semantic reasoning, configuration validation, and expert-level security evaluation.

However, to our knowledge, the intersection of these domains remains unexplored: no existing literature investigates the application of LLMs for analyzing MUD files. This thesis addresses this specific research gap. The subsequent methodology chapter introduces a pipeline design to evaluate whether rule based components augmented with an LLMs can successfully automate the comprehensive pre-deployment validation of MUD policies.

Chapter 4

Methodology and design

4.1 Methodological approach

This chapter defines the methodology used to answer Research Question 2: How could such an analysis be automated with a software? First, we establish the analysis metrics by operationalizing the RFC 8520's requirements for pre-deployment validation. Then we design a modular, five-stage pipeline architecture that combines the existing approaches identified in the related work. And finally, we outline the evaluation strategy for testing the pipeline's effectiveness in analyzing MUD-file security (Q3).

4.2 Analysis metrics

The RFC 8520 recommends that automated systems scan MUD files prior to deployment to ensure they are "both parsable and believable" [11]. While this recommendation sets a necessary baseline, the term "believability" remains conceptually elusive for formal analysis. To bridge this gap, we propose a two-dimensional analysis framework that operationalizes these requirements into measurable metrics.

4.2.1 Dimension 1: Technical Parsability

Parsability serves as the foundational requirement for structural and syntactic conformance. We define a MUD file as technically readable only if it adheres strictly to the JSON serialization of the YANG models defined in RFC 8520 [11]. Since the standard dictates that all extensions must be explicitly declared, we operationalize parsability as the *total absence of schema violations*. This ensures that automated validation algorithms can reliably extract and interpret all defined components without ambiguity.

4.2.2 Dimension 2: From Believability to Accuracy

While parsability is binary, verifying the content of a MUD file is more complex. Foundational data quality research defines believability as the extent to which data is accepted as "true, real, and credible" [29]. However, such a definition is highly subjective and lacks the precision required for a repeatable security analysis.

To resolve this, we pivot from the subjective notion of "believability" to the formal construct of MUD file Accuracy, as established by NIST in their methodology for characterizing IoT network behavior [17]. This shift allows us to decompose the RFC's vague requirement into two objective criteria:

- First, Comprehensiveness: The MUD file must specify all network communications required for the device to function, thereby avoiding security-critical under-specification.
- Second, Correctness: The MUD file must strictly omit unnecessary permissions, thereby avoiding over-specification and adhering to the principle of least privilege.

By substituting the subjective concept of believability with the measurable dimensions of accuracy, we establish a clear defined metric for analyzing MUD files. Our framework evaluates files across both technical (parsability) and functional (accuracy) dimensions, ensuring a holistic assessment of their suitability for secure deployment.

We recognize that this pivot from the RFC's original wording constitutes a deliberate design trade-off. By abandoning the subjective notion of "believability", we inherently lose some of the qualitative flexibility implied by the standard's authors. However, we were honestly suprised that such a vague, unquantifiable term was ever embedded into a formal technical specification in the first place. It may be one of the reasons why no particular solution was standardized within the RFC8520 [11] to mitigate the risk of weaponizing mud files right away. Vice verca, maybe the rather flexible term "believability" was intentionally chosen to allow for a broader set of solutions proposed by the community. By pivoting and applying the strict metrics of correctness and comprehensiveness, we trade subjective nuance for absolute technical clarity, which is implementable in a software solution.

4.2.3 Foundational information security objectives

Crucially, these chosen dimensions fully align with the CIA-Triad, the foundational information security objectives [30]. Specifically, integrity is addressed through parsability,

which confirms that the MUD file is well-formed and structurally sound to guarantee reliable ingestion by the MUD manager. Availability is covered by comprehensiveness, ensuring that all necessary network communications are explicitly declared; this prevents the resulting policy from inadvertently breaking the device's intended functionality. Finally, the framework secures confidentiality by verifying correctness, ensuring that no excessive or unnecessary communications are permitted, thereby minimizing the attack surface and mitigating unauthorized data exposure.

4.3 System design

Rather than inventing a novel validation mechanism from scratch, we deliberately architect a pipeline by combining established frameworks identified in Chapter 3. By synthesizing isolated tools into a single sequential workflow and replacing the human cognitive bottleneck with an LLM, we transform fragmented theoretical concepts into a solid, deployable system. This conservative approach may preclude the development of a completely new, previously unthinkable approach, but it guarantees a rigid, academically proven foundation. To conduct the analysis mapped out in the analysis framework in an automated manner, we design a modular, five-stage prototype pipeline. The following section outlines the key software engineering principles guiding its architecture, followed by a stage-by-stage breakdown of its functional responsibilities.

4.3.1 Design Goals, -Principles and -Decisions

To ensure the pipeline serves its intended purpose we defined explicit design goals, software engineering principles and the central design decision regarding statefulness.

First, we establish three core design goals that the pipeline must satisfy to be effective in real-world industrial SMB environments:

1. Effectiveness: The pipeline must accurately validate MUD files against the established metrics of parsability and functional accuracy.

2. Simplicity and Ease-of-Use: Because the intended end-users are industrial SMBs lacking resources for complex security infrastructures, the system must be inherently simple to deploy and operate.

3. Scalability: The architecture must be capable of handling an increasing number of device profiles without requiring significant structural refactoring.

To meet these goals, the system will be developed in Python, chosen for its widespread adoption, readability, and robust ecosystem for both YANG validation and LLM integration. The implementation is guided by the following design principles:

- Clean Code and PEP 8: The Code strictly adheres to Python's PEP 8 style guide, prioritizing code readability and descriptive naming conventions.
- Atomic Design: Each pipeline stage is encapsulated into an atomic, single-purpose component, ensuring a clear separation of concerns.
- KISS and YAGNI: In alignment with the "Keep It Simple, Stupid" and "You Aren't Gonna Need It" principles, premature optimization and over-engineering were deliberately avoided to maintain a lightweight, auditable codebase.

To satisfy the simplicity constraint required for SMB adoption, the pipeline analyzes each MUD file in strict, stateless isolation. While tools like the MUD-Visualizer demonstrate that aggregating multiple files is theoretically possible, and frameworks such as MUDdy or MUDIS employ complex cross-file similarity analysis, these vectorized architectures require stateful processing and the maintenance of a static vector registry which encodes security invariants or previously classified files.

We judge this stateful complexity to be a significant hurdle for real-world SMB adoption. Operating a central registry or translating own system invariantes into a vector registry introduces significant maintenance overhead that contradicts our explicit design targets. Therefore, we explicitly opt for a purely stateless architecture. We acknowledge that this is a severe design trade-off which shapes the system design, its capabilities and limitations and the final performance. By isolating the analysis, we further deliberately exclude the use of sophisticated interleaved architectures, such as Retrieval-Augmented Generation (RAG) or few-shot prompting in Stage 5 (LLM expert assessment). These contextual architectures could theoretically improve the model's accuracy by referencing past evaluations presented in the Chapter 3 "Related Work". However, we hypothesize that a robust, zero-configuration pipeline will ultimately prove more viable than a marginally more accurate but operationally unmaintainable system. We deliberately sacrifice the analytical depth of cross-file dependencies to align with the KISS principle, ensuring high operational scalability without administrative burden.

4.3.2 Pipeline components

Figure 4.1 illustrates the high-level architecture of the pipeline. Components will be called in sequence, but are implemented as independent modules to enable a modular and extensible design.

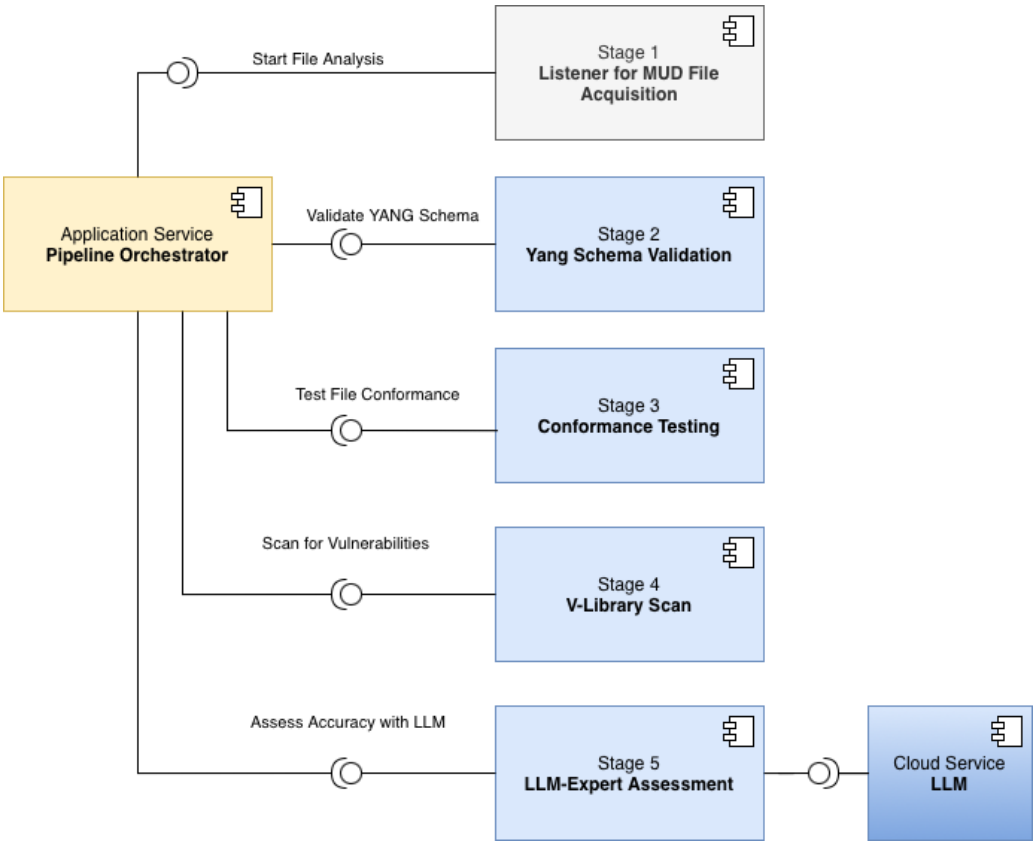

Figure 4.1: Component-level architecture of the pipeline.

Each stage assumes a single, well-defined responsibility grounded in prior research:

The pipeline's control flow is strictly sequential, processing each file through five distinct stages. This ensures that each component can be independently tested, replaced, or extended. The following sections provide a detailed breakdown of the functional responsibilities and design considerations for each stage.

4.3.2.1 Stage 1: Network acquisition and MUD file fetching

Stage 1 simulates the initial device onboarding defined in RFC 8520. It extracts the MUD URL emitted by a newly connected IoT device, retrieves the corresponding MUD file, and forwards it to the analysis pipeline.

In a production environment, this function would be performed by the MUD manager operating directly within the local network infrastructure. For this showcase, Stage 1 acts

Stage	Core Question	Related Work
2 – YANG Validation	Is it a valid MUD YANG model?	MUDdy: YANG validation component
3 – Conformance Testing	How well does it conform?	Systematic Conformance Testing: 26 test-suite
4 – Vulnerability Scan	Does it expose known vulnerable services?	Automated Pre-Certification: V-library
5 – Expert Assessment	Are there miss-specified ACEs?	MUD-Visualizer: Human cognition gap

Table 4.1: Responsibilities and methodological grounding of the pipeline stages.

as a lightweight acquisition proxy to supply the downstream analytical stages. Furthermore, signing and verifying cryptographic MUD file signatures remains out of scope for this prototype. We explicitly opt not to establish a full Public Key Infrastructure (PKI) because it adds no inherent value to the core research questions. Cryptographic signature validation is an already solved engineering problem, extensively standardized in RFC 8520 [11]. By deliberately excluding the signing of MUD files, we simplify the prototype to focus entirely on the novel challenge of MUD file analysis, with the assumption in mind that some files are likely unsigned regardless. Consequently, signature-related conformance tests are simply proxied. The inherent trade-off is that industry adoption requires an extension of the conformance test suite to replace proxied signature tests with full signature validation.

4.3.2.2 Stage 2: YANG schema validation

Stage 2 evaluates the technical parsability of the profile. Following the strict syntactic verification principles established by frameworks like MUDdy [18], it enforces structural compliance with the RFC 8520 YANG models.

Adhering to the fail-fast software engineering principle, Stage 2 serves as the architectural gatekeeper. An invalid schema cannot be meaningfully processed by downstream semantic or logic-based components. Therefore, if a file violates the base syntax, the pipeline halts execution immediately.

4.3.2.3 Stage 3: Systematic conformance testing

Stage 3 quantifies the profile's adherence to the RFC 8520 standard. To establish this metric, it operationalizes the 26-test suite proposed by [12].

The original framework evaluates both device behaviors and file characteristics. Because this thesis isolates the analysis to the MUD file itself, Stage 3 adapts the theoretical framework into a file-centric evaluation. Tests requiring live network state or active device emissions are excluded or proxied, focusing execution exclusively on the file's internal consistency and logical completeness to determine its standardized conformance tier.

The conformance tests will be implemented according to the procedure outlined in the `MUDConformanceTestSpec.pdf` published on GitHub[1] and linked in the original paper from [12].

4.3.2.4 Stage 4: Vulnerability library matching

Stage 4 evaluates functional security adopting the Vulnerability-Library mechanism proposed by [20]. Methodologically, this stage acts as a heuristic blacklist executed against the declared Access Control Entries (ACEs) to identify known vulnerabilities before any semantic analysis occurs. We will use the same V-library test suite as in the original publication. We deliberately positioned this deterministic scan prior to Stage 5 to systematically support the LLM. By executing the blacklist first, we extract hard, rule-based evidence about deprecated protocols or exposed ports. This evidence is then injected as explicit context into the LLM's prompt. This architectural ordering grounds the model's probabilistic reasoning in concrete facts, drastically reducing its cognitive load and mitigating the risk of the AI hallucination.

4.3.2.5 Stage 5: LLM expert assessment

Stage 5 addresses semantic plausibility, which the MUD-Visualizer [21] addresses with visual flow-graphs to bridge the beyond-human-cognition gap. Our design replaces the human-in-the-loop entirely. A Large Language Model, which is designed for large text inputs, will evaluate whether the permitted network flows are logically appropriate for the device type.

Based on the AI integration taxonomy established by [15], the pipeline will implement a reactive LLM architecture. We chose a stateless, single-prompt design over autonomous AI Agents or multi-agent Agentic Systems because the semantic evaluation of a static profile is a triggered, isolated task. It does not require environment perception, external tool execution, or long-horizon planning.

[1] https://github.com/Shuai-Zhang16/MUDConformanceTest/blob/main/MUDConformanceTestSpec.pdf

Architecturally, the LLM is positioned as a supplementary verification layer rather than an absolute authority. It operates under a strict escalate-only policy: the model can never override the status of insecure, based on deterministic vulnerabilities identified during Stage 4.

4.4 Evaluation design

4.4.1 Evaluation strategy

The evaluation will be a qualitative evaluation to assess the feasibility, correctness, and limitations of the pipeline, with a specific focus on the LLM component's decision-making capabilities. It utilizes a baseline collection of MUD files which are subjected to deliberate structural and semantic manipulations. By analyzing both intermediate pipeline outputs and finalized results across these manipulated inputs, the evaluation traces the precise thresholds of failure and success. To isolate the core file analysis logic, the physical network acquisition process (Stage 1) is bypassed, feeding the files directly into the computational pipeline at Stage 2.

4.4.2 Baseline scenario

The experimental baseline scenario is derived from an industrial digital twin starter kit developed by Pepperl+Fuchs in collaboration with Bosch. To simulate a modern, mixed-criticality IoT environment, the original hardware configuration is slightly adjusted. The adapted topology consists of five interconnected devices: wireless vibration sensors, an isolated acoustic sensor, an MQTT bridges,a networked safety blocks, and an edge gateway. This diverse configuration guarantees a comprehensive spectrum of standard, proprietary, and highly restricted communication policies.

4.4.3 Manipulations

We will take the handcrafted MUD files from the baseline scenario collection and manipulate them in various ways to test the limits of the pipeline. These manipulations will be designed to target specific stages of the pipeline, such as introducing schema violations to test Stage 2, designing files which adhere to a specific conformance standard, adding known vulnerable services to test Stage 4, or essentially introducing over- and under-specified ACLs to test the LLM's performance in Stage 5. The specific manipulations and their rationale will be documented in the subsequent chapter.

4.4.4 Ablation testing

To rigorously evaluate the LLM's dependency on explicit semantic cues versus its capability to infer context from raw structural logic, each manipulation scenario incorporates two variants of the MUD file: an altered original and a further stripped version. In the stripped versions, device-identifying strings—such as distinct `system-info` descriptions, explicit URL paths, and descriptive Access Control List (ACL) names—are generalized or removed. For example, an ACL named `from-ipv4-mqtt-bridge` will be renamed to `from-ipv4-device`. We aim to determine whether the LLM's assessment relies on these explicit semantic cues or if it can accurately evaluate the file's functional security based solely on the underlying structural logic and flow patterns.

Chapter 5

Implementation

The Pipeline was implemented according to the design with some minor adjustments. These are documented together with the architectural highlights of each stage in the following sections. The biggest change to the design was that some of the conformance tests were shifted left (to earlier stages), because of a functional overlap. Conformance tests are referred to as "MUD 1.x.y" and align with the enumeration in [12]. Our implementation can be found on GitHub[1]. To run the pipeline, please follow the instructions in the README to set up all the prerequisites. Once completed, the interactive menu (shown in figure 5.1) can be started with `python3 main.py`.

```
 __  __ _   _ ____    _                _           _
|  \/  | | | |  _ \  / \   _ __   __ _| |_   _ ___(_)___
| |\/| | | | | | | |/ _ \ | '_ \ / _` | | | | / __| / __|
| |  | | |_| | |_| / ___ \| | | | (_| | | |_| \__ \ \__ \
|_|  |_|\___/|____/_/   \_\_| |_|\__,_|_|\__, |___/_|___/
                                          |___/

MUDAnalysis Console Menu
Please type 1-5 to start a mode:
1) Pipeline (listener mode)
2) Pipeline (file mode)
3) Evaluate
4) User guide
5) Performance boost
Selection:
```

Figure 5.1: Interactive startup menu of the implementation

[1] https://github.com/vitoxvi/HSG-bachelor-thesis-code

5.1 Architecture overview and output taxonomy

5.1.1 Architecture overview

The implemented architecture exposes one command surface and two explicit runtime flows. The primary entrypoint for manual execution is `python3 main.py`. The menu routes users to the three core command modes

1. `pipeline` (Network listener or file mode),
2. `evaluate` (full batch or selected file), and
3. `help` (renders README-based usage guidance).

In non-interactive execution, `python3 main.py <command>` are directly delegated to `src/app/cli.py` .

Architecturally, the system is separated into three layers with clear responsibilities:

- `Command Layer`: Parses terminal arguments and routes them to the correct mode via `src/app/cli.py`.
- `Application Layer`: Manages the application context (single file vs. batch evaluation) through `src/app/pipeline_service.py` and `src/app/evaluation_service.py`.
- `Domain Layer`: Contains the specific implementation of the five pipeline stages and coordinates their sequential execution via `src/pipeline/runner.py`.

Two runtime flows were implemented. The operational flow, shown in figure 5.2, executes one pipeline run, either from a network announcement (triggered by pipeline stage 1) or from a local file. The evaluation flow executes many pipeline runs over a selected set of files and creates evaluation artifacts (CSV summaries, metrics tables, and one-pager reports per file). Importantly, evaluation does not reimplement pipeline components, but builds on the existing domain logic.

5.1.2 Output taxonomy

To operationalize the conceptual dimensions of “technical parsability” and “accuracy” defined in the methodology, the pipeline first had to establish a strict data taxonomy.

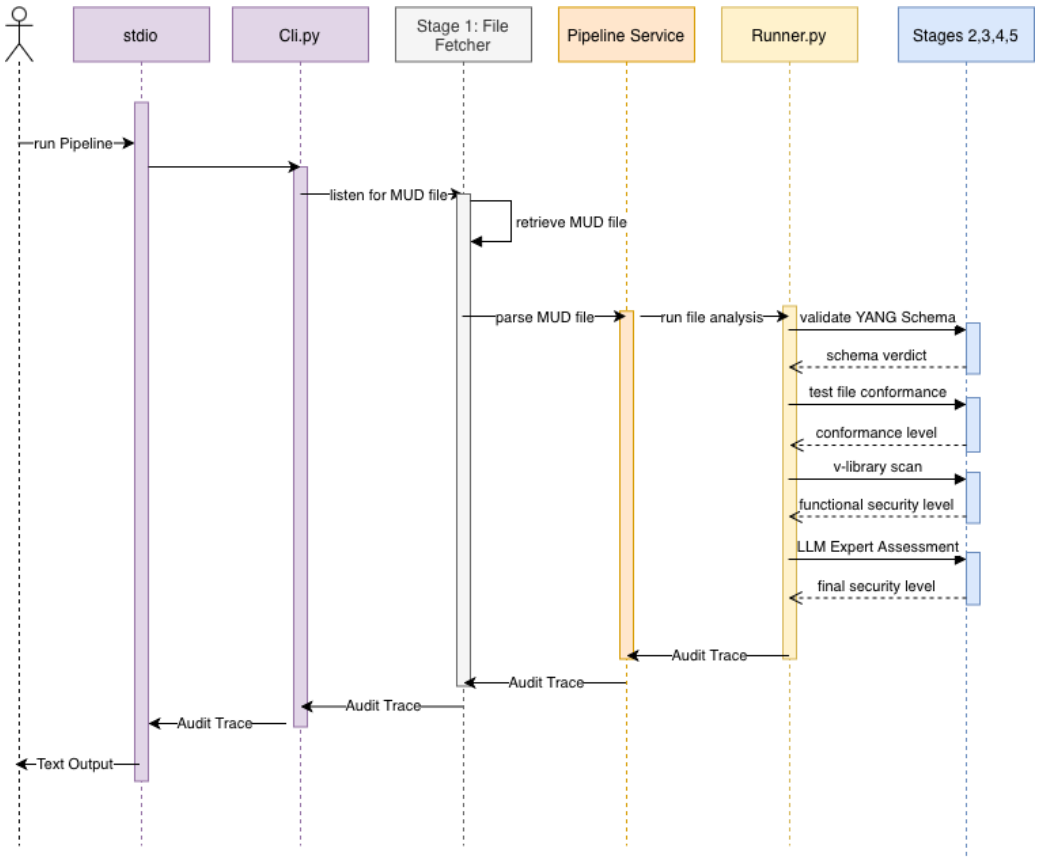

Figure 5.2: Sequence diagram illustrating the execution flow of the pipeline, detailing the interactions between the Command Layer, Application Layer, and Domain Layer during single file processing triggered by the network listener.

To classify "technical parsability", we adopt the hierarchical levels of the conformance testing framework: `Bronze`, `Silver`, `Gold`, and `Platinum` and extend it with a level called `Failed` for files which fail stage 2, the YANG validation.

For the "accuracy" dimension, we build on the V-library output from [20] which uses `OK` and `Vulnearble`, but rename these two levels to `Secure` and `Insecure`, to reflect our broader security assessment scope. Further, `N/A` is assigned if execution aborts early due to parsability level `Failed`.

Consequently, the pipeline generates an independent, two-dimensional assessment for each MUD file, yielding a tuple of `(ConformanceLevel, FunctionalSecurityLevel)`. The output taxonomies are enforced using Python `Literal` types, ensuring static type safety.

5.2 Pipeline implementation

5.2.1 Pipeline execution by the domain service

To strictly separate application-level orchestration (I/O, network listeners, and batch processing) from the core business logic, the pipeline execution is deliberately designed around a *Domain Service* pattern from Domain-Driven Design (DDD).

The application layer handles the outside world:
it normalizes inputs via `run_pipeline_from_file` or `run_pipeline_from_listener` into a standard `AuditInput` object. It then delegates the actual assessment to `audit_mud_file_with_trace(...)` in `src/pipeline/runner.py`. By acting as a Domain Service, the runner is completely oblivious to file systems or network protocols. Instead, it holds the exclusive authority over the audit's domain execution constraints and invariants.

Stage 2 acts as a hard domain gatekeeper: if YANG validation fails, the service short-circuits the pipeline, immediately assigns the `("Failed", "N/A")` taxonomy, and marks downstream stages as skipped. If Stage 2 succeeds, the service orchestrates Stage 3 (conformance), Stage 4 (vulnerabilities), and Stage 5 (expert assessment) sequentially. Stage 5 can only escalate the functional security level from `Secure` to `Insecure`, but it cannot downgrade any previous stage's findings. This design guarantees that the core audit rules remain completely isolated and invariable, whether triggered by a single live network event or a massive background evaluation batch. The Domain Service ultimately aggregates an `AuditTrace` entity, which encapsulates both the intermediate `StageOutcome` records and the final two-dimensional taxonomy pair.

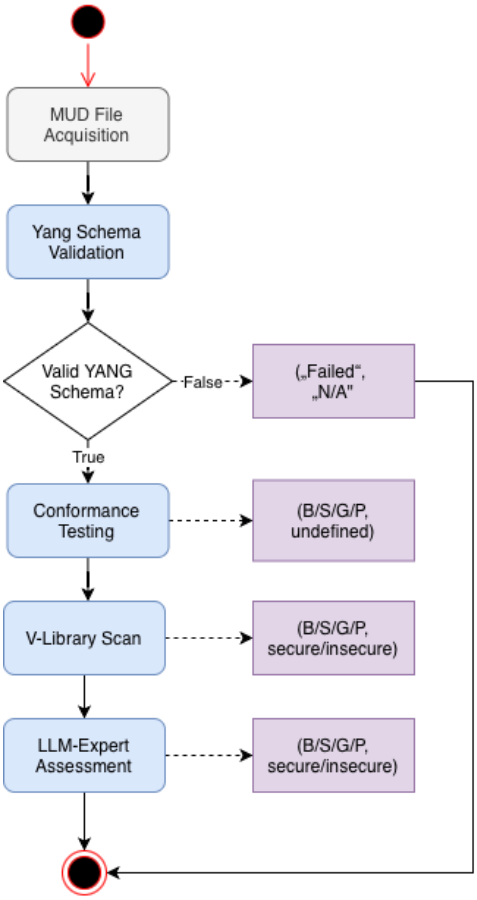

Figure 5.3: Activity diagram illustrating the pipeline execution flow within the domain Service, detailing the sequential processing of stages and the handling of outcomes.

5.2.2 Stage 1: URL announcement and file fetching

The upstream module `stage_fetcher.py` manages network acquisition. It implements a passive listener binding to UDP port 4444 to intercept MUD URL broadcasts, and an active fetcher to download the corresponding MUD files.

The fetcher immediately conducts test MUD.1.2.2 (Verifying MUD file Format) by validating the file's UTF-8 encoding directly after the download. Thereby, any file that successfully passes Stage 1 is guaranteed to be retrievable and correctly encoded before downstream analysis begins.

Moreover, conformance tests MUD 1.1.1 - 1.1.4 (Retrieval and validation of the MUD-URL) are conceptually brought forward into Stage 1. This means we do not actually execute these tests as standalone modules; instead, if Stage 1 successfully fetches the file, we implicitly mark them as passed. We deliberately adopted this architectural deviation to strictly enforce the "shift-left" and "fail-fast" engineering principles. By verifying URL and file accessibility at the absolute edge of the pipeline, we immediately

discard unattainable files, preventing them from consuming computational resources in downstream stages.

In a productive deployment, where Stage 1 would be implemented as part of the MUD Manager, these tests would have to be enforced rigorously.

5.2.3 Stage 2: Schema validation

Stage 2 enforces structural and syntactic compliance using the C-based `yanglint` library, integrated as a stateless `YangModuleValidator` Singleton pattern on module level, which evaluates the payload against nine locally bundled IETF YANG schemas, downloaded from the official IETF repository.

By definition, successfully parsing the profile against the YANG schemas also confirms conformance test MUD.1.2.1 (Verifying MUD format).

5.2.4 Stage 3: Systematic conformance testing

Stage 3 operationalizes the remaining tests of the 26-test suite. The individual tests are instantiated as pure functions, mapping a dictionary cleanly to a boolean verdict, and are decoupled from the execution logic via a central registry. A standalone grading module then sequentially evaluates the registry results to establish the final cumulative grade.

As this implementation omits full cryptographic validation due to the absent PKI, the related tests (MUD 1.1.5 - 1.1.8) are proxied solely by test 1.1.5, which verifies the presence and formatting of the signature field. Thus, if a MUD file contains a signature field ending with ".p7s", tests 1.1.5 - 1.1.8 evaluate to true; otherwise, they evaluate to false. We chose this approach because test 1.1.5 is the only technically feasible check within our reduced setup. Rather than engineering a convoluted, hybrid validation mechanism that deviates from the standard and has no grounding in related work, we deliberately opted to stay strictly within the established conformance framework and proxy the remaining tests. While checking for a mere extension serves as a sufficient structural proxy to validate the pipeline flow in this proof-of-concept, we explicitly acknowledge that this bypass is fundamentally inadequate for any production environment, where strict cryptographic verification is non-negotiable.

Translating the theoretical conformance framework into executable code revealed two apparent misspecifications in the test procedures of two tests:

- MUD.1.4.1 (IP testing on Null-Flags): The test specification suggests validating explicit IP address formats for the `local-networks` field. However, the underlying

standard defines this node as `type empty;`, serving as boolean presence flags rather than a data container. Consequently, it is not defined to carry IP payloads. Our implementation aligns the test procedure with the standard by verifying flag presence rather than attempting to parse non-existent IP data.

- MUD.1.4.2 (URI vs. URL Constraint): The given test specification restricts the `controller` parameter to a network-locatable Uniform Resource Locator (URL). The standard however allows the broader Uniform Resource Identifier (URI) superset, which further supports identifiers such as URNs. Our implementation relaxes this constraint, employing the `rfc3986` library to execute standard-compliant URI validation instead of URL-specific checks.

These points of divergence suggest that the framework of [12] may require further refinement regarding tests MUD 1.4.1 and MUD 1.4.2.

5.2.5 Stage 4: V-Library testing

Stage 4 implements the vulnerability library from [20], adopting four of the five original tests (VT-1 through VT-4). The TLS version check (VT-5) is excluded because standard MUD profiles operate strictly at network Layers 3 and 4. We opted to drop this check because specifying application-layer cryptography would require non-standard extensions beyond the base RFC 8520 specification. While Gangurde's original framework could seamlessly evaluate TLS versions by monitoring live network traffic [20], our static analysis pipeline operates exclusively on the abstract MUD file where such runtime details simply do not exist. Relying on an extension to the standard would add additional complexity which contradicts our explicit design targets. As a trade-off we lose a valid check for secure communication, which should be enforced in a production environment.

To decouple the assessment logic from the deeply nested MUD JSON schema, the system employs a dedicated extraction utility, which iterates over the `ietf-access-control-list` nodes once and flattens them into a standardized sequence of immutable `ServiceDescription` dataclasses, similar to [20]. The vulnerability scanner subsequently evaluates these objects using a fail-fast heuristic: if any service triggers a rule match, the pipeline immediately assigns the `Insecure` status.

5.2.6 Stage 5: LLM-based analysis

As per design decision, the AI component was implemented as a reactive, zero-shot prompt-based LLM call. This architecture, is heavily dependent on the selected model

and the prompt design. Thus, these were the main focus points during the implementation of this stage: translating the findings and key takeaways from [22] into a concrete, operationalized prompt structure and enabling a flexible backend that can be easily switched between different LLM providers.

The prompt design utilizes the CO-STAR (Context, Objective, Style, Tone, Audience, Response) framework [31]. This baseline is augmented with prompt engineering best-practices recommended in Claude's documentation [32]. We integrate explicit role assignment, XML-like sectioning, and placing large context blocks at the beginning into our prompt design.

We must explicitly acknowledge at this point that engineering the "perfect" prompt is practically impossible. Development in Generative AI outpaces academic validation, leaving formal research sparse while social platforms like X, Reddit, and LinkedIn overflow with announcements of new "groundbreaking" techniques. We deliberately opted for CO-STAR because it has proven its practical efficacy by winning a major prompt engineering competition in 2024 [33]. While it may not be the absolute cutting edge, it aligns with our design strategy of conservatively relying on evidence-based and established frameworks.

To transition from general text generation to a strict decision-taking workflow, the CO-STAR elements are adapted. The *Style* parameter is replaced with precise *Audit Guidelines* to govern the decision process. The *Audience* parameter is substituted with a *Responsibility* definition to enforce accountability. The final prompt structure is composed of the following sections:

1. <Role> </Role>
2. <Context> </Context>
3. <Objective> </Objective>
4. <Audit Guidelines> </Audit Guidelines>
5. <Tone> </Tone>
6. <Responsibility> </Responsibility>
7. <Response> </Response>

Context fields are dynamically populated using available MUD file metadata and upstream stage results. This selective inclusion minimizes token volume and prevents

```
<Context>
    <Device Info>
        <Manufacturer>{manufacturer_if_available}</Manufacturer>
        <Device Systeminfo>{systeminfo_if_available}</Device Systeminfo>
        <Is supported>{is_supported_if_available}</Is supported>
        <Device Model Name>{model_name_if_available}</Device Model Name>
        <Declared ACLs>
            - {acl_line_1}
            - {acl_line_2}
            - ...
        </Declared ACLs>
    </Device Info>

    [...]

    <Previous Stages>
        A previous rule-based check against a vulnerability-library has
    determined that this file specifies {no_vulnerabilities|
    known_vulnerabilities|an_uncertain_vulnerability_state}.
        The file {was|was_not} validly signed and {does|does_not} contain
    all optional metadata fields.
        Stage-4 flagged ACEs:
        - {stage4_match_summary_1}
        - {stage4_match_summary_2}
        - ...
    </Previous Stages>
</Context>
```

Figure 5.4: Context section of the prompt blueprint with dynamic placeholders for MUD metadata and upstream stage results.

context dilution. We reason that enclosing all extracted MUD fields strictly within explicit XML-like delimiters provides the LLM with clear structural guidance. By dynamically injecting only available fields and presenting them in a standardized, aggregated form, we expect to drastically reduce noisy context. This focused presentation should empower the model to concentrate entirely on semantic plausibility rather than parsing raw JSON syntax.

The dynamic context block is presented in Figure 5.4. The full prompt blueprint is available in the Appendix A.

At the software architecture level, the backend employs a modulare design, to decouple the LLM call from APIs. A single configuration variable allows seamless runtime

toggling between different API adapters (e.g offline models via `ollama` or cloud-native APIs like `gemini`).

5.3 Evaluation implementation

To execute the evaluation strategy defined in the methodology, an independent automated testing setup was developed. This setup directly feeds manipulated MUD files into the analysis pipeline to test its boundaries and edge cases. The following sections describe the architecture of this setup and the implementation of the experimental dataset.

5.3.1 Automated evaluation architecture

The evaluation logic is isolated in a dedicated `evaluation` directory, separating experimental orchestration from the main application layer. The evaluation suite is built around three core architectural principles to ensure reproducibility, scale, and resilience.

Instead of hardcoding test conditions in Python, the suite relies on a `scenario_registry.json` file. This file acts as a single source of truth, mapping each tested MUD file to its expected results and detailing how it was manipulated. To add new test cases, one simply appends an entry to this JSON file and links it to the additional file.

The evaluation script calls the exact same `runner.py` domain service used for regular operations. This ensures that the evaluation authentically reflects how the tool performs in reality.

Testing deliberately corrupted MUD files inevitably triggers exceptions. The batch runner catches and isolates these crashes to prevent experiment termination. If a file fails critically, the error is safely caught and the suite advances to the next file.

To support the subsequent data analysis, the pipeline continuously generates execution artifacts during this batch process. Each evaluated MUD file yields a detailed execution trace, which is immediately merged with the expected ground-truth metadata from the `scenario_registry.json`. This combined data is written to a dedicated Markdown one-pager report inside the `one-pagers/` directory. The file documents the expected versus actual taxonomy outcomes, execution durations, and serialized intermediate JSON outputs for every pipeline stage.

Simultaneously, the pipeline accumulates core result parameters in memory. Upon batch completion, these results are written in bulk to a central `[timestamp]-evaluation.csv` file. A separate `[timestamp]-metrics-summary.csv` calculates and exports overall performance metrics, such as accuracy and precision.

5.3.2 Experimental dataset implementation

The baseline MUD file collection was constructed representing the devices and topological constraints defined in the methodology. For each of the five core devices (Vibration Sensor, Acoustic Sensor, MQTT-Bridge, Edge Gateway, and Safety Block), a compliant, valid baseline MUD file was handcrafted. The full content of these baseline MUD files can be found in Appendix B.

To systematically execute the diverse edge-cases detailed in the methodology, the baseline files were structurally manipulated. These targeted violations, spanning syntactic errors, specific conformance configurations, injected legacy ports, and deliberate ACL misconfigurations, are stored as independent JSON MUD files. The complete `scenario_registry.json`, linking manipulation scenarios to files, is documented in Appendix C.

To support the planned ablation study, two physical variants were generated for every scenario configuration: one with and one without hints on the device's type.

Chapter 6

Evaluation

This chapter evaluates the implemented pipeline to systematically answer Research Question 3: How effectively can the resulting software analyze security in MUD files?

6.1 Evaluation setup and methodology

6.1.1 Evaluation setup

The evaluation employs a qualitative, exploratory methodology. Rather than running the original collection of manipulated files only once, MUD files were iteratively modified again and again to expose the pipeline's boundaries and decision logic. After each run, evaluation artifacts were examined to understand the pipeline's reasoning and guide the design of subsequent test cases. We chose this dynamic probing over a static test set because the behavior of LLMs is inherently probabilistic and difficult to map with binary pass/fail metrics. While this exploratory approach trades off strict methodological reproducibility for qualitative depth, we consider it essential. It provides the only realistic mechanism to genuinely uncover how the model reacts to subtle semantic shifts, prompt variations, and unforeseen edge cases, which we were not able to predict in advance.

6.1.2 Preliminary runs

Before conducting the main evaluation, three preliminary runs were conducted using the open-source model `Qwen2.5:7b` deployed locally via Ollama on a MacBook Air M2 16GB. These runs served to identify and correct deficiencies in the prompt before the main evaluation.

Three specific weaknesses were discovered and addressed:

1. The initial prompt caused all files to be classified as *Insecure*, with reasoning that any communication exceeded strict Least Privilege requirements. The prompt was revised to balance Least Privilege with functional necessity.

2. In the second iteration, where metadata fields were missing, the model struggled with "N/A" placeholders, generating false-positive concerns. The prompt was corrected to instruct the model to omit missing fields entirely rather than process null values.

3. In the third iteration, the model misunderstood MUD's whitelist semantics, flagging empty ACL blocks as over-permissive. A dedicated "MUD Logic Constraints" section was added to explain whitelist semantics.

After these three iterations, prompt tuning was halted to prevent overfitting to limited test data.

6.1.3 Main runs

Four main evaluation runs were conducted using `Gemini 2.5 Pro` via the Google AI Studio API. The test runs and modifications took place as follows:

- Run 1 (Baseline Manipulations): Established the baseline detection capabilities against a standard set of manipulated IoT device profiles.

- Run 2 (No MQTT / Safety): The initial run revealed that legitimate but unencrypted MQTT traffic and open Safety Block connections heavily influenced the LLM's risk assessments. Consequently, these connections were removed or upgraded to MQTTS across the Gateway and Bridge. This isolated the core anomalies, testing whether the LLM could still identify threats with fewer obvious security flaws.

- Run 3 (Active Obfuscation): This iteration simulated an active Red-Team operation. Access Control Entry (ACE) names were systematically relabeled with benign terminology to camouflage the injected vulnerabilities and evaluate the model's resilience against semantic deception.

- Run 4 (No V-Lib): To determine the LLM's standalone capabilities, the deterministic V-Library checks for standard vulnerabilities (e.g., Ports 80 and 22) were disabled.

- Extra Run (No V-Lib): To further test the LLM's ability to detect dangerous configurations without the V-Library, two new MUD files were created with an open Port 80, one bound to `local-networks` and one without any target.

Else, pipeline stages remained unchanged from the documented implementation. All generated evaluation artifacts are archived in the project's GitHub repository together with the input MUD files per run.

6.1.4 Qualitative evaluation metrics

Quantitative metrics of precision and recall were strictly and solely applied to the technical readability classification (Bronze, Silver, Gold, Platinum). An example output of run 1 is shown in Table 6.2

Conversely, counting binary *Secure vs. Insecure* assessments for the LLM outcomes proved misleading. Such metrics conflate coincidental correct classifications with correct reasoning. For example, a file might be correctly flagged as *Insecure*, but for an entirely hallucinated or unrelated reason.

To address this, semantic accuracy was measured instead: verifying whether the pipeline correctly identified the *specific critical manipulation* embedded in each test file, regardless of its final binary classification. This decoupled threat detection from the classification labels and provided a clear, accurate signal about the LLM's reasoning boundaries.

Table 6.1: Detection accuracy of injected MUD file manipulations across evaluation runs. An 'X' indicates the targeted vulnerability was successfully identified by the LLM.

Manipulations	**Run 1**	**Run 2**	**Run 3**	**Run 4**	**Extra Run**
Test Setup	Baseline Manip.	No MQTT/ Safety	Active Obfuscation	No V-Lib (08, 09)	New Files (http det.)
http downgrade	X	X	X	X	
left ssh port	X	X	X	X	
missing internet connection				X	
half a standard	X	X			
missing DNS					
open https port	X	X		LLM API Error	
silent data exfil	X	X		LLM API Error	
implausible connection	X	X		LLM API Error	
lateral movement pivot	X	X	X	X	
http local bound w/o v-library					X
http global w/o v-library					X

6.2 Evaluation results

The sections below present observed results aggregated across runs, followed by consolidated findings.

6.2.1 Baseline scenarios

Baseline files (derived from the Digital Twin Starter Kit) were generally evaluated correctly. Gemini 2.5 Pro successfully inferred plausible device profiles and communication patterns from MUD content. Qwen2.5:7b, by contrast, classified nearly all files as Insecure without meaningful discrimination.

In Run 1, the Edge Gateway was flagged Insecure due an unencrypted MQTT and an unencrypted Modbus connections. While these choices were intentional in the device design, they represent valid concerns for the LLM. It rationalizes the decision explicitly:

> *"The device profile is functionally over-permissive and miss-specified, granting a cloud-connected AI gateway active control over a safety-critical relay via the insecure Modbus protocol. Additionally, it permits unencrypted local MQTT communication, which violates the principle of defense-in-depth."*

All five baseline files were correctly assigned Conformance Level Platinum.

In Run 2, MQTT connections were upgraded to MQTTS and the Safety Block link was removed. The Edge Gateway was then correctly reclassified as Secure. However, the MQTT Bridge was newly flagged Insecure because it accepted MQTTS upstream but forwarded unencrypted MQTT downstream—a valid asymmetry observation by the model.

6.2.2 Syntax and conformance testing

Syntax validation (Stage 2) detected all intentional format errors consistently and at the correct line. Conformance classification (Stage 3) reliably categorized files across all four levels (Bronze, Silver, Gold, Platinum).

Files with deliberate format violations were rejected early as "Failed, N/A." A Bronze-level file lacking any ACLs was correctly labeled "functionally insufficient and therefore insecure" by the LLM. Deterministic stages (Stages 1–3) performed without error across all test runs.

Table 6.2: Evaluations Metric of the Pipeline in Run 1 (Conformance Level)

metric_group	positive_label	elib.	TP	FP	TN	FN	Prec.	Rec.	F1	Dur.
r1_conf_level	Failed	38	4	0	34	0	1	1	1	19.04
r1_conf_level	Silver	38	1	0	37	0	1	1	1	19.04
r1_conf_level	Gold	38	16	0	22	0	1	1	1	19.04
r1_conf_level	Platinum	38	16	0	22	0	1	1	1	19.04
r1_conf_level	**OVERALL_EXACT**	**38**	**38**	**0**	**0**	**0**	**1**	**1**	**1**	**19.04**

6.2.3 Under-specification

The pipeline performed poorly on detecting under-specified MUD files. The LLM struggled to identify missing functional rules. For example, an Edge Gateway deliberately stripped of necessary internet access and DNS rules was never flagged as under-permissive across any run.

Detection of partial standard implementations was highly erratic. The Precision Time Protocol (PTP) strictly requires both UDP/319 and UDP/320. Specifying only half of the standard is inherently flawed regardless of the device context. Yet, the LLM identified the missing UDP/320 port in only two out of eight iterations (Runs 1 and 2 with device-info):

> *"However, the profile is functionally deficient due to an incomplete PTP specification. The omission of the standard PTP general message port (UDP/320) constitutes a miss-specification that could impair the device's critical time-sync."*

In all other iterations—including Run 3 (with full device context) and the ablation runs (without semantic hints)—the exact same broken ruleset was falsely verified as complete. Instead of flagging the protocol error, the model actively justified as sufficient:

> *"The declared ACLs for MQTT telemetry and PTP time synchronization are necessary and sufficient for a device in a time-sensitive control system."*

This highlights a significant vulnerability of the Pipeline: Simple rule-based components such as the V-library are not able to detect under-specification, because they evaluate declared rules and cannot audit what is absent. Moreover, the LLM fails to consistently apply objective protocol knowledge, showing that its detection of under-specification is fundamentally unreliable.

6.2.4 Over-specification

Conversely, the pipeline reliably identified over-specified rules and correctly named the associated threat patterns. It consistently detected unnecessary Port 80 (HTTP downgrade) and Port 22 (SSH) ACEs, even with the V-Library disabled.

The LLM successfully identified complex threat scenarios. In Run 1, it flagged a DNS tunnel as a *"high-risk covert channel for potential data exfiltration."* Similarly, it correctly associated an anomalous SMB rule with its specific threat mechanism, noting that *"SMB is a common protocol exploited for lateral movement by malware."* The model also reliably recognized implausible external connections, such as a Safety Block communicating over the internet in Run 2:

> *"The device's declared ACLs are functionally insecure due to over-permissiveness. They permit communication with an external IP address for remote control, which violates the Principle of Least Privilege for a safety-critical component and introduces an unacceptable remote attack surface."*

6.2.5 Obfuscation

While the pipeline successfully detected overt threats, active obfuscation (renaming Access Control Entries (ACEs) to mimic legitimate functions) partially bypassed the LLM's detection logic.

Semantic deception effectively masked several critical vulnerabilities. A wide-open HTTPS (Port 443) rule renamed from `https-any` to `local-web-dashboard` was no longer flagged as dangerous, even though it was not bound to the `local-networks` target. The LLM was heavily influenced by the ACE name, ignoring the underlying lack of target binding shown in the ACE 6.1:

Rather than flagging this unbound port as a critical vulnerability, the model justified the configuration based entirely on semantic trust:

> *"The declared ACLs for the Industrial Edge AI Gateway are semantically secure and functionally aligned with its role. The device's communication footprint [...] adheres to the Principle of Least Privilege without any indication of over-permissiveness or functional deficiency."*

Similarly, the previously detected data exfiltration DNS tunnel and the implausible Safety Block remote connection were interpreted as valid safety features when relabeled as `remote-emergency-connection`.

```
{
  "name": "to-local-web-dashboard",
  "matches": {
    "ipv4": { "protocol": 6 },
    "tcp": {
      "destination-port": { "operator": "eq", "port": 443 },
      "ietf-mud:direction-initiated": "to-device"
    }
  },
  "actions": { "forwarding": "accept" }
}
```

Figure 6.1: ACE named "local-web-dashboard", but matching any connection of protocol 6 (TCP) with no bound target.

In the Red-Team scenario (Run 3), the model's heavy reliance on ACE names became evident. However, obfuscation did not bypass all protocols. An SSH connection disguised as a `vendor-diagnosis-tunnel` was correctly recognized and flagged as a potential pivot point. Likewise, lateral movement via SMB, relabeled as `scada history sync`, was accurately rejected as unnecessary and functionally excessive:

> *"The device profile is deemed insecure due to two critical flaws. Firstly, it declares the use of the insecure SMB protocol (TCP/445) for data synchronization, which represents an excessive and unnecessary security risk."*

6.2.6 Impact of semantic hints

While obfuscation actively introduces deceptive semantics, comparing the paired file variants (with and without `system-info` and telling names) across the runs evaluated the passive removal of context.

For the specified scenarios, removing these semantic hints had minimal effect on threat detection. The LLM simply generated vaguer device profiles, deriving a generic "IoT Field Device" rather than a specific device type, but still maintained similar detection capabilities.

6.2.7 V-Library contribution

As previously noted in the over-specification results, run 4 and the extra run confirmed that the LLM reliably identifies common protocol vulnerabilities (e.g., HTTP over Port

80, SSH over Port 22) independently. This finding isolates baseline LLM capability and should not be interpreted as a replacement for deterministic safeguards.

When evaluated in isolation, the current minimal V-Library reliably flags the few services it explicitly encodes. However, unknown or non-encoded manipulations, such as the obfuscated HTTPS backdoor scenario or the labeled SMB scenario, were not picked up by this rule set. In scenarios without Port 80 or 22 patterns, the V-Library frequently classified files as secure, confirming that its present coverage is narrow.

Chapter 7

Discussion

7.1 The duality of LLM capabilities

The evaluation shows a clear split between deterministic rules and probabilistic AI. The rule-based stages (syntax and conformance) were exceptionally reliable, providing a solid baseline. In contrast, the LLM was fluid and probabilistic. It reasoned through complex intents that rigid systems missed, but it had contextual blind spots.

We explicitly designed a hybrid pipeline to leverage the strengths of both approaches. The advantage of this design is that we combine strict rule enforcement with flexible semantic reasoning. The downside is that we inherit the weaknesses of both: the LLM cannot replace the rule-based system, and the rule-based system cannot guide the LLM when it lacks context. This contrast proves why our hybrid architecture is essential: neither system can safely operate alone.

The evaluation reveals a strict asymmetry in the LLM's analytical capabilities. The model excels at conceptual threat modeling but lacks rigorous protocol-level grounding. Consequently, it identifies over-specification effectively. By inferring a plausible device profile, the LLM accurately detects superfluous access control rules and articulates the associated threat vectors surprisingly precisely.

Conversely, the model performs poorly when analyzing under-specification. It consistently fails to recognize missing essential communication rules, such as incomplete protocol-port pairings. Furthermore, the analysis exhibits deterministic inconsistencies. The LLM occasionally hallucinates threat rationales for functional deficits or misinterprets complex network topologies.

We attribute this to the model's lack of external ground-truth knowledge. While the LLM perfectly reads the provided context, it has no reference detailing what a specific proprietary IoT device *must* do. Without this baseline, it cannot tell a deliberately lean

security posture from a broken, under-specified one. It was an intended design choise to keep the LLM architecture without such an external baseline (such as in a RAG or few-shot prompt), which now shows to be a limitation. Ultimately, we classify this outcome as a partial success: the pipeline effectively flags security risks, but it fails to guarantee device operability.

7.2 Semantic deception

The LLM seems to trust descriptive text more than objective structural rules. If the text contradicts the network logic, the LLM favors the text. Adversaries can exploit this simply by adopting trusted names.

This vulnerability became obvious when we contrasted passive context removal with active deception. When we passively removed device identifiers, the LLM remained stable. It inferred a generic profile and flagged excessive rules based purely on structural topology. However, when an adversary actively injected benign-sounding labels over malicious rules, the model's reasoning was hijacked. It abandoned structural logic and rationalized the malicious traffic as a legitimate feature.

This danger reinforces the value of human-in-the-loop systems found in related work. By trying to fully automate human judgment with an LLM, we inadvertently introduce a vulnerability that human visual inspection naturally prevents. The MUD Visualizer [21] approach relies on pruned and merged visual graphs, and ACE naming is therefore discarded during the judgement process. Thus, while replacing human cognition with LLMs is structurally feasible, it extends the threat surface with an additional vulnerability to mitigate.

7.3 Deterministic safeguards

To provide real value and counteract semantic deception, deterministic safeguards must move beyond a minimal signature library like our V-Library. The V-Library works for the few patterns it knows, but it cannot reason about structurally misleading rules. In the methodology, we deliberately chose not to implement mathematically encoded invariants (like those proposed by MUDdy [18]) to keep the system simple and avoid organizational overhead. After evaluating the results, we must personally classify this trade-off as too optimistic. The added complexity of stateful invariants is strictly necessary to provide robust safety against semantic manipulation.

7.4 Key takeaways from the evaluation

The evaluation highlights the distinct strengths and boundaries of combining deterministic validation with LLM-based semantic analysis. The observed results converge into five key takeaways:

Deterministic reliability. The early pipeline stages (Syntax and Conformance validation) proved highly robust. Format violations were consistently caught early, and MUD files were accurately categorized into their respective conformance levels without fail.

Detection asymmetry. The pipeline exhibits a strong asymmetry in anomaly recognition. It is highly capable of identifying over-specification (superfluous ports, lateral movement, data exfiltration) and accurately attributing threat models. Conversely, it systematically fails to detect under-specification, frequently ignoring the absence of essential rules like required DNS or gateway internet access.

Vulnerability to semantic deception. The LLM's analytical capability is highly dependent on ACE nomenclature. Active obfuscation severely degrades detection accuracy. When attackers mask malicious rules with benign-sounding names, the LLM not only fails to detect the threat but often actively rationalizes the injected rules as legitimate system features.

Context Dependency. Passive removal of semantic hints (device information and descriptive names) does not significantly impair misspecification detection. However, active obfuscation that manipulates these semantic cues can significantly increase the likelihood of bypassing detection.

V-Library coverage and expansion. The No V-Lib runs show that the LLM can detect very common vulnerabilities (e.g., wide-open Ports 80 and 22) in isolation. This is likely because such well known patterns do often occur in the training data. However, this does not remove the need for deterministic safeguards. The current rule-based layer remains too narrow and should be expanded from a small signature set toward broader structural invariants.

7.5 Limitations

While the preceding sections detail specific analytical failures, they point toward several systemic limitations defining the boundaries of our implementation.

Absence of device-specific ground truth The pipeline's primary constraint is its lack of a reference model for device behavior. Operating in a functional vacuum, it can critique

what is present but cannot effectively audit what is absent. Without a deterministic library to compare profiles against, the pipeline remains a risk-reduction tool rather than a guarantee of functional integrity. By intentionally opting for a stateless architecture in our methodology to ensure administrative simplicity, we traded away rigorous cross-contextual validation. While this choice kept the pipeline easy to deploy, we must now recognize this trade-off as too costly for a productive solution. Future work must bridge this gap by exploring lightweight, hybrid state models that encode critical invariants without forcing administrators to manually maintain exhaustive sets of rules.

Adversarial sensitivity The pipeline relies too heavily on the honesty of the input's metadata. Because semantic analysis prioritizes linguistic cues (ACE names) over structural logic, it lacks adversarial resilience. Our architecture treats the LLM as a "trusted final assessor," creating a single point of failure where a successful semantic obfuscation bypasses the entire security assessment. This highlights the need for a robust "defense-in-depth" approach where multiple components tightly validate syntax and semantics before allowing the LLM to act as a final aggregator. This single point of failure might could have been mitigated by deploying an Agentic AI System rather than a reactive LLM. In an orchestrated, multi-agent design, a specialized "challenger agent" could actively challenge the primary LLM's assessments to detect obfuscation. However, we stand by our architectural decision. An AI agent or multi-agent system is over-engineered for a task specific, single-file analysis. Adopting such an architecture would introduce severe latency, skyrocket token costs, and have violated our core design principle of simplicity for SMB adoption.

Scalability and infrastructure dependency The evaluation reveals a steep performance ceiling related to model scaling. The significant disparity between the local `Qwen2.5:7b` and the API-based `Gemini 2.5 Pro` proves that the pipeline's utility is tied to resource-intensive models. During preliminary testing, the local Qwen model repeatedly hallucinated context and struggled with strict prompt formatting, making the pipeline highly unstable. Only the cloud-native Gemini model possessed the reasoning density to execute qualitative judgments consistently. This introduces a critical trade-off: local, privacy-preserving deployments are highly desirable for industrial SMBs protecting operational technology (OT) networks, but these local models currently lack the necessary logic capabilities. Conversely, forcing reliance on high-performing cloud APIs introduces network latency, external dependencies, increased cost and severe data privacy

risks. Future adaptors will have to weigh the cost and benefits of using local models vs. cloud-native APIs.

Restricted rule-based components. The deterministic components are limited by their reliance on a predefined rule set. While effective for known vulnerabilities, this approach cannot adapt to novel attack vectors. As discussed in the methodology, we deliberately placed the V-Library before the LLM to ground its reasoning in hard facts. However, the static nature of the V-Library means it delivers additional evidence only in very specific cases.

7.6 Reflection on the core design goals

In light of these accumulated limitations, we must critically re-examine the core design goals that have guided us throughout this work: effectiveness, simplicity, and scalability. In hindsight, our strict adherence to simplicity and effortless scalability, which were driven by the desire to accommodate resource-constrained small and medium-sized businesses (SMBs), may have been a strategic misstep. Perhaps we should have prioritized sophistication and a thorough analysis, and accepted the harsh reality that in the modern cybersecurity landscape, "simple and easy-to-maintain" is no longer sufficient. Ultimately, even SMBs must prioritize resources and radically improve their security ecosystems. An approach focused on building a sophisticated security ecosystem would have enabled the integration of more complex, stateful components that could have mitigated many of the critical limitations discovered in this work.

7.7 Operationalizing the pipeline: A proposed redesign

Given the limitations we identified above, we conclude that LLMs must not operate as autonomous judges for zero-touch network onboarding. The model is simply too vulnerable to active semantic manipulation. Instead, we propose redesigning the pipeline to enforce more sophisticated rules while still not relying on peripheral systems such as vectorized MUD profiles or mathematical invariants.

To operationalize this framework safely, we propose extending the rule-based component across three invariant classes: completeness checks, structural checks, and endpoint plausibility. We could implement this by explicitly evaluating:

- Mandatory communication invariants for critical protocols (e.g., flagging the absence of DNS reachability) to mechanically detect under-specification.

- Access Control Entries (ACEs) lacking explicit target bindings, ensuring no rule defaults to dangerous wide-open states.
- The presence of hardcoded IP addresses, forcing manufacturers to rely on resolvable domain names.
- Automated WHOIS resolutions for all external endpoints to verify target legitimacy prior to semantic evaluation.
- Strict regular expressions for naming conventions, deliberately rejecting ambiguous names (like "any" or "open") and enforcing that "local" targets map exclusively to local subnets.

By augmenting the deterministic layer with these invariants, we propose a robust yet simple safety net which should catch the structural manipulations and omission-based weaknesses that the LLM consistently overlooked. However, this is of course tailored to the exact limitations our evaluation revealed. Ultimately, it will be the responsibility of established solution developers (such as the CISO or NIST) to strike a balance between simplicity and strict security enforcement.

Finally, transitioning this architecture into a production deployment requires translating our two-dimensional output matrix into actionable onboarding policies. We envision organizations mapping this matrix directly to their specific risk appetite. For instance, an risk-averse environment might configure the MUD manager to automatically onboard only `[Platinum, Secure]` files, while mandating manual reviews for `[Gold, Secure]` profiles, and instantly discarding any file flagged as `[Failed]`. Ultimately, we judge that while the pipeline provides the analytical engine, translating "technical parsability" and "accuracy" into definitive onboarding rules remains a strategic, organizational decision.

Chapter 8

Future work

After evaluating the results of the implemented architecture, we were left with two questions which sparked our curiosity. First, given that our automated pipeline is vulnerable to semantic deception, would a human expert actually perform any better under the same conditions? E.g. would a human expert not be deceived by an unbound HTTP rule called "Web Dashboard" and reliably flag it as a critical vulnerability? And second, since we deliberately stripped away complex stateful architectures to prioritize administrative simplicity, could reintegrating these sophisticated designs patch the vulnerabilities we discovered? The following sections outline how future research could systematically address these two open questions.

8.1 Human versus machine vulnerability studies

Future research should benchmark LLM security assessments against human domain experts. Research could expand the MUD-Visualizer experiment [22] by including an LLM test group to establish a comparative baseline. The original study demonstrated that humans require visual aids to comprehend complex MUD profiles. Comparing an LLM processing raw text against a human evaluating visualizations answers two fundamental questions: Is only an LLM susceptible to semantic obfuscation and does it process text reliably enough to outperform visually aided experts?

8.2 Hybrid validation: Bridging static and dynamic analysis

Future architectures could couple static onboarding pipelines, such as presented in this thesis, with continuous dynamic runtime monitoring, such as presented for example by [20], to pick up under-specified files. This would not require to invent a completely new approach but be be a bit less conservative with the components chosen and integrated into the system. Integrating runtime analyses, which compare specified MUD Profiles

to exhibited communication patterns, would enable the identification of functional gaps dynamically by continuously monitoring the network.

Chapter 9

Conclusion

The Internet of Things is experiencing unprecedented growth, driving a necessary shift toward Zero Trust architectures. The Manufacturer Usage Description (MUD) standard facilitates this transition by introducing micro-segmentation in IoT. However, it inherently introduces the critical vulnerability of "blind onboarding." By automatically applying manufacturer-provided policies, local networks delegate their security perimeters to third parties without verifying the actual implications of those rules. This thesis evaluated an automated software solutions to mitigate this risk specifically addressing the resource constraints of industrial SMBs in Switzerland.

Addressing the first research question (How can MUD files be analyzed for security risks before deployment?) reveals that MUD files can be analyzed for security risks before deployment by combining existing validation approaches. As detailed in Chapter 3 "Related work", existing tools address the problem from distinct mathematical, conformance, and visual analysis angles. We evaluated this research landscape and concluded it is sophisticated but fragmented. Structural frameworks such as isolated conformance tiers effectively validate protocol syntax, while other tools assess plausibility and cross-context consistency. We judged mathematical models like MUDdy as formally rigorous but operationally prohibitive. We found visual frameworks like MUD-Visualizer effective for human intuition but unscalable for automation. Taken together, they demonstrate that pre-deployment analysis is possible, but not yet unified.

Regarding the second research question (How could such an analysis be automated with software?), the implemented system proves that this process can be fully automated through a custom pipeline architecture. As engineered in Chapter 4 "Methodology and design", we first had to resolve the RFC 8520's subjective demand for "believability" by explicitly operationalizing it into measurable dimensions of technical parsability and functional accuracy. To execute this framework, we architected a modular, five-stage

workflow that synthesizes YANG validation, conformance testing, vulnerability scanning, and a reactive Large Language Model assessment. We explicitly opted for a stateless design to guarantee simplicity and ease-of-use for resource-constrained SMBs. However, hindsight reveals that prioritizing stateless scalability misaligned with the reality of complex network threats; discarding cross-device context severely limits the detection of sophisticated manipulations. Nevertheless, the shows the feasibility of connecting these existing tools into a single workflow and translates these fragmented checks into an actionable, two-dimensional output taxonomy.

Answering the third research question (How effectively can the resulting software analyze security in MUD files?), the evaluation reveals a strict duality in analysis effectiveness. We found that deterministic, rule-based components provide an exceptionally reliable baseline for structural integrity, proving that syntax and protocol conformance can be mechanically guaranteed. However, this narrow rule coverage is fundamentally insufficient as a standalone defense. Conversely, we demonstrated that Large Language Models excel at contextual threat detection; they successfully infer plausible device profiles to accurately identify over-specification risks. This proves that AI can effectively replace human intuition for spotting excessive permissions. Yet, this capability is severely bottlenecked by two critical flaws. First, we observed a strict detection asymmetry: the model consistently fails to identify functional under-specification because it lacks an external ground-truth for specific device requirements. Second, we discovered a profound vulnerability to active semantic deception. When adversaries mask malicious rules with benign-sounding labels, the LLM abandons structural logic and actively rationalizes the injected threats. The primary takeaway is that while generative AI successfully flags complex anomalies, its reliance on linguistic cues introduces novel attack vectors. Consequently, the resulting software operates effectively as a sophisticated risk-reduction engine, but it cannot serve as an autonomous, infallible judge.

Ultimately, our work proves that the critical vulnerability of "blind onboarding" introduced in Chapter 1 “Introduction” can be mitigated, but not by blindly trusting a new black box. We demonstrated that generative Artificial Intelligence is not yet fit to act as a fully autonomous judge for network defense; its vulnerability to semantic deception undermines the very Zero Trust principles the MUD standard is meant to enforce. While our pipeline successfully evaluates the true implications of manufacturer-provided policies, we conclude that replacing human intuition entirely with AI merely trades one attack vector for another. We judge this not as a failure of the concept, but as a

necessary architectural pivot for a next iteration build. The path forward is clear for us: future deployments must anchor AI-driven anomaly detection within rigorous, stateful mathematical invariants. While there is certainly still work to be done, we are proud of the foundation built here. By mapping out where deterministic rules succeed and where AI currently fails, this thesis takes a real, practical step toward solving the problem of blind onboarding. Addressing this problem was a highly motivating challenge, and we hope our findings provide a solid starting point for the next generation of automated MUD file analysis tools.

Postscriptum: In praxi

For industrial SMBs navigating this landscape today, our final engineering recommendation is cautious adoption and to prioritize resources. Before any fully autonomous tool can be deployed, a human-in-the-loop approach must remain the default. Although our pipeline revealed significant limitations with our concrete design, there are already many frameworks that can pragmatically be linked together today. A highly effective baseline that mitigates most immediate risks could be formed by combining strict cryptographic signature validation, requiring Platinum conformance, and visual manual expert review via MUD-Visualizer. However, for absolute safety in today's threat landscape, a sophisticated set of interconnected, stateful components must be constructed according to one's resources and requirements.

Bibliography

[1] Jameel Shehu Yalli, Mohd Hilmi Hasan, and Aisha Abubakar Badawi. "Internet of Things (IoT): Origins, Embedded Technologies, Smart Applications, and Its Growth in the Last Decade". In: *IEEE Access* 12 (2024), pp. 91357–91382. DOI: 10.1109/ACCESS.2024.3418995.

[2] Mohammed Aziz Al Kabir, Wael Elmedany, and Mhd Saeed Sharif. "Securing IoT Devices Against Emerging Security Threats: Challenges and Mitigation Techniques". In: *Journal of Cyber Security Technology* 7.4 (2023), pp. 199–223. DOI: 10.1080/23742917.2023.2228053.

[3] Lena Boeckmann. *Challenges of IoT Security*. Project Report. Hamburg University of Applied Sciences (HAW Hamburg), 2023. URL: https://inet.haw-hamburg.de/teaching/ws-2022-23/project-class/fw1_lena_boeckmann.pdf (visited on 02/28/2026).

[4] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. "Understanding the Mirai Botnet". In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017, pp. 1093–1110. ISBN: 978-1-931971-40-0. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis.

[5] Kaushik Ragothaman, Yong Wang, Bhaskar Rimal, and Mark Lawrence. "Access Control for IoT: A Survey of Existing Research, Dynamic Policies and Future Directions". In: *Sensors* 23.4 (2023). ISSN: 1424-8220. DOI: 10.3390/s23041805.

[6] Mujtaba Awan and Abu Alam. "Cybersecurity Threats and Defensive Strategies for Small and Medium Firms: A Systematic Mapping Study". In: *Administrative Sciences* 15.12 (2025). ISSN: 2076-3387. DOI: 10.3390/admsci15120481.

[7] Schweizerische Eidgenossenschaft: KMU.admin.ch. *Figures on SMEs: Companies and jobs*. Nov. 2025. URL: https://www.kmu.admin.ch/kmu/en/home/concrete-know-how/facts-and-figures/figures-smes/companies-and-jobs.html (visited on 04/08/2026).

[8] Naeem Firdous Syed, Syed W. Shah, Arash Shaghaghi, Adnan Anwar, Zubair Baig, and Robin Doss. "Zero Trust Architecture (ZTA): A Comprehensive Survey". In: *IEEE Access* 10 (2022), pp. 57143–57179. DOI: 10.1109/ACCESS.2022.3174679.

[9] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. *Zero Trust Architecture*. NIST Special Publication 800-207. National Institute of Standards and Technology, 11, 2020. DOI: 10.6028/NIST.SP.800-207.

[10] Donna Dodson, Douglas Montgomery, Tim Polk, Mudumbai Ranganathan, Murugiah Souppaya, Steve Johnson, Ashwini Kadam, Craig Pratt, Darshak Thakore, Mark Walker, Eliot Lear, Brian Weis, William C. Barker, Dean Coclin, Avesta Hojjati, Clint Wilson, Tim Jones, Adnan Baykal, Drew Cohen, Kevin Yeich, Yemi Fashina, Parisa Grayeli, Joshua Harrington, Joshua Klosterman, Blaine Mulugeta, Susan Symington, and Jaideep Singh. *Securing Small-Business and Home Internet of Things (IoT) Devices: Mitigating Network-Based Attacks Using Manufacturer*

Usage Description (MUD). NIST Special Publication 1800-15. National Institute of Standards and Technology (NIST), 26, 2021. DOI: 10.6028/NIST.SP.1800-15.

[11] Eliot Lear, Ralph Droms, and Dan Romascanu. *Manufacturer Usage Description Specification*. RFC 8520. Mar. 2019. DOI: 10.17487/RFC8520.

[12] Shuai Zhang, Hugo Sullivan, Krishna Kumar Lahoti, and Hassan Habibi Gharakheili. "Systematic Verification of IoT Device Conformance to Ietf Manufacturer Usage Description Standard". In: *IEEE Communications Standards Magazine* 9.1 (2025), pp. 59–66. DOI: 10.1109/MCOMSTD.0001.2400051.

[13] European Parliament and Council of the European Union. *Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act)*. Official Journal of the European Union, OJ L, 12 July 2024. 2024. URL: https://eur-lex.europa.eu/eli/reg/2024/1689/oj (visited on 03/23/2026).

[14] N Breznau and HHV Nguyen. "An Introduction to Generative Artificial Intelligence for Academics [version 2; peer review: 2 approved]". In: *F1000Research* 14.655 (2026). DOI: 10.12688/f1000research.166513.2.

[15] Ranjan Sapkota, Konstantinos I. Roumeliotis, and Manoj Karkee. "AI Agents vs. Agentic AI: A Conceptual taxonomy, applications and challenges". In: *Information Fusion* 126 (2026), p. 103599. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2025.103599.

[16] José L. Hernández-Ramos, Sara N. Matheu, Angelo Feraudo, Gianmarco Baldini, Jorge Bernal Bernabe, Poonam Yadav, Antonio Skarmeta, and Paolo Bellavista. "Defining the Behavior of IoT Devices Through the MUD Standard: Review, Challenges, and Research Directions". In: *IEEE Access* 9 (2021), pp. 126265–126285. DOI: 10.1109/ACCESS.2021.3111477.

[17] Paul Watrobski, Murugiah Souppaya, Joshua Klosterman, and William C. Barker. *Methodology for Characterizing Network Behavior of Internet of Things Devices*. NIST IR 8349. Gaithersburg, MD, Aug. 28, 2025. DOI: 10.6028/NIST.IR.8349. published.

[18] Ayyoob Hamza, Dinesha Ranathunga, Hassan Habibi Gharakheili, Matthew Roughan, and Vijay Sivaraman. "Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles". In: *Proceedings of the 2018 Workshop on IoT Security and Privacy*. 2018, pp. 8–14. ISBN: 9781450359054. DOI: 10.1145/3229565.3229566.

[19] Anat Bremler-Barr, Bar Meyuhas, and Ran Shister. "MUDIS: MUD Inspection System". In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. 2022, pp. 1–3. DOI: 10.1109/NOMS54207.2022.9789891.

[20] C.A. Gangurde. *Automation of IoT Pre-Certification Security Testing Environment Based on the Manufacturing Usage Description*. Master Thesis. Eindhoven University of Technology, May 27, 2019. URL: https://pure.tue.nl/ws/portalfiles/portal/130181666/Thesis_Report_Chaitali_Gangurde_1177108.pdf.

[21] Vafa Andalibi, Eliot Lear, DongInn Kim, and L. Jean Camp. *On the Analysis of MUD-Files' Interactions, Conflicts, and Configuration Requirements Before Deployment*. 2021. URL: https://arxiv.org/abs/2107.06372.

[22] Vafa Andalibi, Jayati Dev, DongInn Kim, Eliot Lear, and L. Jean Camp. "Is Visualization Enough? Evaluating the Efficacy of MUD-Visualizer in Enabling Ease of Deployment for Manufacturer Usage Description (MUD)". In: *Proceedings of the 37th Annual Computer Security Applications Conference*. 2021, pp. 337–348. ISBN: 9781450385794. DOI: 10.1145/3485832.3485879.

[23] Ruiqi Wang, Jiyu Guo, Cuiyun Gao, Guodong Fan, Chun Yong Chong, and Xin Xia. "Can LLMs Replace Human Evaluators? An Empirical Study of LLM-as-a-Judge in Software Engineering". In: *Proceedings of the ACM on Software Engineering* 2.ISSTA (2025), pp. 1955–1977. ISSN: 2994-970X. DOI: 10.1145/3728963.

[24] Abdulla Almenhali. *Retrieval-Augmented LLM Analysis of IoT Source Code for Automatic MUD Profile Support*. Master Thesis. University of Padova, 2025. URL: https://thesis.unipd.it/retrieve/38384164-bef6-4c38-9c66-50a6a93ab7e6/AbdullaAlmenhali_Cybersecurity_MsC_Thesis__UniPD.pdf (visited on 03/03/2026).

[25] Ziyang Ye, Triet Huynh Minh Le, and M. Ali Babar. *LLMSecConfig: An LLM-Based Approach for Fixing Software Container Misconfigurations*. 2025. arXiv: 2502.02009 [cs.SE]. URL: https://arxiv.org/abs/2502.02009.

[26] Xinyu Lian, Yinfang Chen, Runxiang Cheng, Jie Huang, Parth Thakkar, Minjia Zhang, and Tianyin Xu. "Large Language Models as Configuration Validators". In: *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. 2025, pp. 1704–1716. DOI: 10.1109/ICSE55347.2025.00017.

[27] Dipkamal Bhusal, Md Tanvirul Alam, Le Nguyen, Ashim Mahara, Zachary Lightcap, Rodney Frazier, Romy Fieblinger, Grace Long Torales, Benjamin A. Blakely, and Nidhi Rastogi. *SECURE: Benchmarking Large Language Models for Cybersecurity*. 2024. arXiv: 2405.20441 [cs.CR]. URL: https://arxiv.org/abs/2405.20441.

[28] Fikret Mert Gultekin, Oscar Lilja, Ranim Khojah, Rebekka Wohlrab, Marvin Damschen, and Mazen Mohamad. *Leveraging Large Language Models for Cybersecurity Risk Assessment – A Case from Forestry Cyber-Physical Systems*. 2025. arXiv: 2510.06343 [cs.SE]. URL: https://arxiv.org/abs/2510.06343.

[29] Richard Y. Wang and Diane M. Strong. "Beyond Accuracy: What Data Quality Means to Data Consumers". In: *Journal of Management Information Systems* 12.4 (1996), pp. 5–33. ISSN: 0742-1222. DOI: 10.1080/07421222.1996.11518099.

[30] National Institute of Standards and Technology (NIST). *Standards for Security Categorization of Federal Information and Information Systems*. Tech. rep. FIPS PUB 199. U.S. Department of Commerce, 2004. DOI: 10.6028/NIST.FIPS.199.

[31] Nzubechukwu C. Ohalete, Kevin B. Gittner, and Lauren M. Matheny. *COSTAR-A: A prompting framework for enhancing Large Language Model performance on Point-of-View questions*. 2025. arXiv: 2510.12637 [cs.CL]. URL: https://arxiv.org/abs/2510.12637.

[32] *Prompting Best Practices*. Claude API Docs. URL: https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/claude-prompting-best-practices (visited on 04/03/2026).

[33] Sheila Teo. *How I Won Singapore's GPT-4 Prompt Engineering Competition*. Towards Data Science. 27, 2024. URL: https://towardsdatascience.com/how-i-won-singapores-gpt-4-prompt-engineering-competition-34c195a93d41 (visited on 04/07/2026).

Appendix A
Prompt blueprint

```
1.0.1 {
1.0.2 <Role>
1.0.3     You are a Senior Specialist in Industrial IoT Systems performing
        semantic security assessment of declared device communication before
        their onboarding.
1.0.4 </Role>
1.0.5
1.0.6 <Context>
1.0.7     <Device Info>
1.0.8         <Manufacturer>{manufacturer_if_available}</Manufacturer>
1.0.9         <Device Systeminfo>{systeminfo_if_available}</Device Systeminfo>
1.0.10        <Is supported>{is_supported_if_available}</Is supported>
1.0.11        <Device Model Name>{model_name_if_available}</Device Model Name>
1.0.12        <Declared ACLs>
1.0.13            - {acl_line_1}
1.0.14            - {acl_line_2}
1.0.15            - ...
1.0.16        </Declared ACLs>
1.0.17    </Device Info>
1.0.18
1.0.19    <MUD Logic Constraints>
1.0.20        WHITELIST_MODEL: MUD is a strict Default-Deny architecture.
        Absence of ACEs within an ACL means no communication allowed.
1.0.21        RECIPROCAL INTEGRITY: MUD files use a reciprocal model. For every
        'from-device' (Egress) flow, a corresponding 'to-device' (Ingress)
        return-path ACE must exist with inverted ports.
1.0.22        ABSTRACTIONS: "controller", "manufacturer", and "my-controller"
        are verified network abstractions. Treat them as specific network
        entities, not vague placeholders.
1.0.23    </MUD Logic Constraints>
1.0.24
1.0.25    <Previous Stages>
```

```
1.0.26          A previous rule-based check against a vulnerability-library has
        determined that this file specifies {no_vulnerabilities|
        known_vulnerabilities|an_uncertain_vulnerability_state}.
1.0.27          The file {was|was_not} validly signed and {does|does_not} contain
        all optional metadata fields.
1.0.28          Stage-4 flagged ACEs:
1.0.29          - {stage4_match_summary_1}
1.0.30          - {stage4_match_summary_2}
1.0.31          - ...
1.0.32      </Previous Stages>
1.0.33 </Context>
1.0.34
1.0.35 <Objective>
1.0.36      Perform a Semantic Security Audit. Synthesize a device profile from
        the provided metadata (if available) and audit the ACLs for '
        Functional Alignment'.  A file is accurate if, the declared behavior
        is consistentwith the device's role. Detect over-permissiveness, under-
        permissiveness or miss-specification for the device role. Your goal is
         to ensure the ACL footprint is consistent with the device's
        operational role while maintaining the Principle of Least Privilege.
1.0.37 </Objective>
1.0.38
1.0.39 <audit_guidelines>
1.0.40      "audit_guidelines": [
1.0.41          "1. Functional Necessity (Access Excess): Identify ACEs that
        violate the Principle of Least Privilege. Flag as 'Insecure' any
        protocol, port, or endpoint_class that is not plausibly required for
        the device's primary role.",
1.0.42          "2. Functional Sufficiency (Access Deficiency): Identify missing
        connectivity. Flag as 'Insecure' if the ACLs omit protocols or ports
        essential for the device to perform its declared function.",
1.0.43          "3. Behavioral Profile Consistency (Holistic Alignment):
        Synthesize all ACEs into a single 'Behavioral Footprint'. Evaluate if
        this collective footprint is contextually plausible for the devices
        role. Flag as 'Insecure' if the device exhibits 'Identity Bloat' or
        attempts to interact with out-of-domain systems that do not align with
         its primary function.",
1.0.44          "4. Procedural Constraints: Ground every verdict in explicit
        evidence from ACL and ACE fields. As the Stage 5 gatekeeper, you may
        escalate functional security severity based on semantic context, but
        you are strictly forbidden from downgrading previous stage findings."
```

```
    ]
</audit_guidelines>

<Tone>
    Formal, precise, and evidence-based.
</Tone>

<Responsibility>
    You are the final gatekeeper for MUD file onboarding. Your assessment
    directly impacts whether a device is allowed to operate in the network.
     Your decision must be justified with clear reasoning based on the
    ACLs and security principles.
</Responsibility>

<Response>
    Return exactly one JSON object and no extra text.
    {
        "proposed_functional_security_level": "Secure|Insecure|N/A",
        "reasoning": "A high-level executive summary (2-3 sentences)
    explaining the semantic alignment between the device role and the
    declared ACLs.",
        "evidence_chain": [
            "ACE-grounded evidence items"
        ]
    }
</Response>
}
```

Appendix B
Baseline MUD profiles

B.1 Baseline scenario MUD files

This appendix contains the raw JSON source code for the baseline MUD files used during the evaluation in Chapter 6.

B.1.1 Acoustic sensor

```
{
  {
  "ietf-mud:mud": {
    "mud-version": 1,
    "mud-url": "https://our-factory.com/mud/acoustic-sensor.json",
    "last-update": "2026-03-24T09:15:00+01:00",
    "mud-signature": "https://our-factory.com/mud/general-signature.p7s",
    "cache-validity": 40,
    "is-supported": true,
    "systeminfo": "Isolated Acoustic Sensor (Profile-Driven)",
    "from-device-policy": {
      "access-lists": {
        "access-list": [
          {
            "name": "from-ipv4-acoustic"
          }
        ]
      }
    },
    "to-device-policy": {
      "access-lists": {
        "access-list": [
          {
            "name": "to-ipv4-acoustic"
          }
        ]
      }
```

```
2.0.28      }
2.0.29    },
2.0.30    "ietf-access-control-list:acls": {
2.0.31      "acl": [
2.0.32        {
2.0.33          "name": "from-ipv4-acoustic",
2.0.34          "type": "ietf-access-control-list:ipv4-acl-type",
2.0.35          "aces": {
2.0.36            "ace": [
2.0.37              {
2.0.38                "name": "from-coap-audio-stream",
2.0.39                "matches": {
2.0.40                  "ietf-mud:mud": {
2.0.41                    "my-controller": [null]
2.0.42                  },
2.0.43                  "ipv4": {
2.0.44                    "protocol": 17
2.0.45                  },
2.0.46                  "udp": {
2.0.47                    "destination-port": {
2.0.48                      "operator": "eq",
2.0.49                      "port": 5683
2.0.50                    }
2.0.51                  }
2.0.52                },
2.0.53                "actions": {
2.0.54                  "forwarding": "accept"
2.0.55                }
2.0.56              }
2.0.57            ]
2.0.58          }
2.0.59        },
2.0.60        {
2.0.61          "name": "to-ipv4-acoustic",
2.0.62          "type": "ietf-access-control-list:ipv4-acl-type",
2.0.63          "aces": {
2.0.64            "ace": [
2.0.65              {
2.0.66                "name": "to-coap-audio-stream-reply",
2.0.67                "matches": {
2.0.68                  "ietf-mud:mud": {
```

```
2.0.69                  "my-controller": [null]
2.0.70                },
2.0.71                "ipv4": {
2.0.72                  "protocol": 17
2.0.73                },
2.0.74                "udp": {
2.0.75                  "source-port": {
2.0.76                    "operator": "eq",
2.0.77                    "port": 5683
2.0.78                  }
2.0.79                }
2.0.80              },
2.0.81              "actions": {
2.0.82                "forwarding": "accept"
2.0.83              }
2.0.84            }
2.0.85          ]
2.0.86        }
2.0.87      }
2.0.88    ]
2.0.89  }
2.0.90 }
2.0.91 }
```

B.1.2 Vibration sensor

```
 2.0.1 {
 2.0.2   {
 2.0.3   "ietf-mud:mud": {
 2.0.4     "mud-version": 1,
 2.0.5     "mud-url": "https://our-factory.com/mud/vibration-sensor.json",
 2.0.6     "last-update": "2026-03-24T09:45:00+01:00",
 2.0.7     "mud-signature": "https://our-factory.com/mud/general-signature.p7s",
 2.0.8     "cache-validity": 40,
 2.0.9     "is-supported": true,
2.0.10     "systeminfo": "Condition Monitoring Vibration Sensor",
2.0.11     "from-device-policy": {
2.0.12       "access-lists": {
2.0.13         "access-list": [
2.0.14           {
```

```
2.0.15            "name": "from-ipv4-vibration"
2.0.16          }
2.0.17        ]
2.0.18      }
2.0.19    },
2.0.20    "to-device-policy": {
2.0.21      "access-lists": {
2.0.22        "access-list": [
2.0.23          {
2.0.24            "name": "to-ipv4-vibration"
2.0.25          }
2.0.26        ]
2.0.27      }
2.0.28    }
2.0.29  },
2.0.30  "ietf-access-control-list:acls": {
2.0.31    "acl": [
2.0.32      {
2.0.33        "name": "from-ipv4-vibration",
2.0.34        "type": "ietf-access-control-list:ipv4-acl-type",
2.0.35        "aces": {
2.0.36          "ace": [
2.0.37            {
2.0.38              "name": "from-mqtt-telemetry",
2.0.39              "matches": {
2.0.40                "ietf-mud:mud": {
2.0.41                  "my-controller": [null]
2.0.42                },
2.0.43                "ipv4": {
2.0.44                  "protocol": 6
2.0.45                },
2.0.46                "tcp": {
2.0.47                  "destination-port": {
2.0.48                    "operator": "eq",
2.0.49                    "port": 1883
2.0.50                  },
2.0.51                  "ietf-mud:direction-initiated": "from-device"
2.0.52                }
2.0.53              },
2.0.54              "actions": {
2.0.55                "forwarding": "accept"
```

```
2.0.56              }
2.0.57            },
2.0.58            {
2.0.59              "name": "from-ptp-sync-event",
2.0.60              "matches": {
2.0.61                "ietf-mud:mud": {
2.0.62                  "same-manufacturer": [null]
2.0.63                },
2.0.64                "ipv4": {
2.0.65                  "protocol": 17
2.0.66                },
2.0.67                "udp": {
2.0.68                  "destination-port": {
2.0.69                    "operator": "eq",
2.0.70                    "port": 319
2.0.71                  }
2.0.72                }
2.0.73              },
2.0.74              "actions": {
2.0.75                "forwarding": "accept"
2.0.76              }
2.0.77            },
2.0.78            {
2.0.79              "name": "from-ptp-sync-gen",
2.0.80              "matches": {
2.0.81                "ietf-mud:mud": {
2.0.82                  "same-manufacturer": [null]
2.0.83                },
2.0.84                "ipv4": {
2.0.85                  "protocol": 17
2.0.86                },
2.0.87                "udp": {
2.0.88                  "destination-port": {
2.0.89                    "operator": "eq",
2.0.90                    "port": 320
2.0.91                  }
2.0.92                }
2.0.93              },
2.0.94              "actions": {
2.0.95                "forwarding": "accept"
2.0.96              }
```

```
2.0.97              },
 2.0.98              {
 2.0.99                "name": "from-ptp-sync-event-reply",
2.0.100                "matches": {
2.0.101                  "ietf-mud:mud": {
2.0.102                    "same-manufacturer": [null]
2.0.103                  },
2.0.104                  "ipv4": {
2.0.105                    "protocol": 17
2.0.106                  },
2.0.107                  "udp": {
2.0.108                    "source-port": {
2.0.109                      "operator": "eq",
2.0.110                      "port": 319
2.0.111                    }
2.0.112                  }
2.0.113                },
2.0.114                "actions": {
2.0.115                  "forwarding": "accept"
2.0.116                }
2.0.117              },
2.0.118              {
2.0.119                "name": "from-ptp-sync-gen-reply",
2.0.120                "matches": {
2.0.121                  "ietf-mud:mud": {
2.0.122                    "same-manufacturer": [null]
2.0.123                  },
2.0.124                  "ipv4": {
2.0.125                    "protocol": 17
2.0.126                  },
2.0.127                  "udp": {
2.0.128                    "source-port": {
2.0.129                      "operator": "eq",
2.0.130                      "port": 320
2.0.131                    }
2.0.132                  }
2.0.133                },
2.0.134                "actions": {
2.0.135                  "forwarding": "accept"
2.0.136                }
2.0.137              }
```

```
2.0.138           ]
2.0.139         }
2.0.140       },
2.0.141       {
2.0.142         "name": "to-ipv4-vibration",
2.0.143         "type": "ietf-access-control-list:ipv4-acl-type",
2.0.144         "aces": {
2.0.145           "ace": [
2.0.146             {
2.0.147               "name": "to-ptp-sync-event",
2.0.148               "matches": {
2.0.149                 "ietf-mud:mud": {
2.0.150                   "same-manufacturer": [null]
2.0.151                 },
2.0.152                 "ipv4": {
2.0.153                   "protocol": 17
2.0.154                 },
2.0.155                 "udp": {
2.0.156                   "destination-port": {
2.0.157                     "operator": "eq",
2.0.158                     "port": 319
2.0.159                   }
2.0.160                 }
2.0.161               },
2.0.162               "actions": {
2.0.163                 "forwarding": "accept"
2.0.164               }
2.0.165             },
2.0.166             {
2.0.167               "name": "to-ptp-sync-gen",
2.0.168               "matches": {
2.0.169                 "ietf-mud:mud": {
2.0.170                   "same-manufacturer": [null]
2.0.171                 },
2.0.172                 "ipv4": {
2.0.173                   "protocol": 17
2.0.174                 },
2.0.175                 "udp": {
2.0.176                   "destination-port": {
2.0.177                     "operator": "eq",
2.0.178                     "port": 320
```

```
2.0.179                 }
2.0.180               }
2.0.181             },
2.0.182             "actions": {
2.0.183               "forwarding": "accept"
2.0.184             }
2.0.185           },
2.0.186           {
2.0.187             "name": "to-mqtt-telemetry-reply",
2.0.188             "matches": {
2.0.189               "ietf-mud:mud": {
2.0.190                 "my-controller": [null]
2.0.191               },
2.0.192               "ipv4": {
2.0.193                 "protocol": 6
2.0.194               },
2.0.195               "tcp": {
2.0.196                 "source-port": {
2.0.197                   "operator": "eq",
2.0.198                   "port": 1883
2.0.199                 }
2.0.200               }
2.0.201             },
2.0.202             "actions": {
2.0.203               "forwarding": "accept"
2.0.204             }
2.0.205           },
2.0.206           {
2.0.207             "name": "to-ptp-sync-event-reply",
2.0.208             "matches": {
2.0.209               "ietf-mud:mud": {
2.0.210                 "same-manufacturer": [null]
2.0.211               },
2.0.212               "ipv4": {
2.0.213                 "protocol": 17
2.0.214               },
2.0.215               "udp": {
2.0.216                 "source-port": {
2.0.217                   "operator": "eq",
2.0.218                   "port": 319
2.0.219                 }
```

```
2.0.220                 }
2.0.221               },
2.0.222               "actions": {
2.0.223                 "forwarding": "accept"
2.0.224               }
2.0.225             },
2.0.226             {
2.0.227               "name": "to-ptp-sync-gen-reply",
2.0.228               "matches": {
2.0.229                 "ietf-mud:mud": {
2.0.230                   "same-manufacturer": [null]
2.0.231                 },
2.0.232                 "ipv4": {
2.0.233                   "protocol": 17
2.0.234                 },
2.0.235                 "udp": {
2.0.236                   "source-port": {
2.0.237                     "operator": "eq",
2.0.238                     "port": 320
2.0.239                   }
2.0.240                 }
2.0.241               },
2.0.242               "actions": {
2.0.243                 "forwarding": "accept"
2.0.244               }
2.0.245             }
2.0.246           ]
2.0.247         }
2.0.248       }
2.0.249     ]
2.0.250   }
2.0.251 }
2.0.252 }
2.0.253
```

B.1.3 MQTT bridge

```
 2.0.1 {
 2.0.2   {
 2.0.3   "ietf-mud:mud": {
```

```
2.0.4     "mud-version": 1,
2.0.5     "mud-url": "https://our-factory.com/mud/mqtt-bridge-master.json",
2.0.6     "last-update": "2026-03-22T11:46:00+01:00",
2.0.7     "mud-signature": "https://our-factory.com/mud/general-signature.p7s",
2.0.8     "cache-validity": 40,
2.0.9     "is-supported": true,
2.0.10    "systeminfo": "MQTT Bridge Mesh Aggregator",
2.0.11    "from-device-policy": {
2.0.12      "access-lists": {
2.0.13        "access-list": [
2.0.14          {
2.0.15            "name": "from-ipv4-mqtt-bridge"
2.0.16          }
2.0.17        ]
2.0.18      }
2.0.19    },
2.0.20    "to-device-policy": {
2.0.21      "access-lists": {
2.0.22        "access-list": [
2.0.23          {
2.0.24            "name": "to-ipv4-mqtt-bridge"
2.0.25          }
2.0.26        ]
2.0.27      }
2.0.28    }
2.0.29  },
2.0.30  "ietf-access-control-list:acls": {
2.0.31    "acl": [
2.0.32      {
2.0.33        "name": "from-ipv4-mqtt-bridge",
2.0.34        "type": "ipv4-acl-type",
2.0.35        "aces": {
2.0.36          "ace": [
2.0.37            {
2.0.38              "name": "from-mqtt-upstream",
2.0.39              "matches": {
2.0.40                "ietf-mud:mud": {
2.0.41                  "my-controller": [null]
2.0.42                },
2.0.43                "ipv4": {
2.0.44                  "protocol": 6
```

```
2.0.45                },
2.0.46                "tcp": {
2.0.47                  "destination-port": {
2.0.48                    "operator": "eq",
2.0.49                    "port": 1883
2.0.50                  },
2.0.51                  "ietf-mud:direction-initiated": "from-device"
2.0.52                }
2.0.53              },
2.0.54              "actions": {
2.0.55                "forwarding": "accept"
2.0.56              }
2.0.57            },
2.0.58            {
2.0.59              "name": "from-mqtt-downstream-reply",
2.0.60              "matches": {
2.0.61                "ietf-mud:mud": {
2.0.62                  "controller": "urn:local:sensor:vibration"
2.0.63                },
2.0.64                "ipv4": {
2.0.65                  "protocol": 6
2.0.66                },
2.0.67                "tcp": {
2.0.68                  "source-port": {
2.0.69                    "operator": "eq",
2.0.70                    "port": 1883
2.0.71                  }
2.0.72                }
2.0.73              },
2.0.74              "actions": {
2.0.75                "forwarding": "accept"
2.0.76              }
2.0.77            }
2.0.78          ]
2.0.79        }
2.0.80      },
2.0.81      {
2.0.82        "name": "to-ipv4-mqtt-bridge",
2.0.83        "type": "ipv4-acl-type",
2.0.84        "aces": {
2.0.85          "ace": [
```

```
2.0.86              {
2.0.87                "name": "to-mqtt-downstream",
2.0.88                "matches": {
2.0.89                  "ietf-mud:mud": {
2.0.90                    "controller": "urn:local:sensor:vibration"
2.0.91                  },
2.0.92                  "ipv4": {
2.0.93                    "protocol": 6
2.0.94                  },
2.0.95                  "tcp": {
2.0.96                    "destination-port": {
2.0.97                      "operator": "eq",
2.0.98                      "port": 1883
2.0.99                    },
2.0.100                   "ietf-mud:direction-initiated": "to-device"
2.0.101                 }
2.0.102               },
2.0.103               "actions": {
2.0.104                 "forwarding": "accept"
2.0.105               }
2.0.106             },
2.0.107             {
2.0.108               "name": "to-mqtt-upstream-reply",
2.0.109               "matches": {
2.0.110                 "ietf-mud:mud": {
2.0.111                   "my-controller": [null]
2.0.112                 },
2.0.113                 "ipv4": {
2.0.114                   "protocol": 6
2.0.115                 },
2.0.116                 "tcp": {
2.0.117                   "source-port": {
2.0.118                     "operator": "eq",
2.0.119                     "port": 1883
2.0.120                   }
2.0.121                 }
2.0.122               },
2.0.123               "actions": {
2.0.124                 "forwarding": "accept"
2.0.125               }
2.0.126             }
```

```
2.0.127           ]
2.0.128         }
2.0.129       }
2.0.130     ]
2.0.131   }
2.0.132 }
2.0.133 }
```

B.1.4 Edge gateway

```
 2.0.1 {
 2.0.2   {
 2.0.3   "ietf-mud:mud": {
 2.0.4     "mud-version": 1,
 2.0.5     "mud-url": "https://our-factory.com/mud/edge-gateway.json",
 2.0.6     "last-update": "2026-03-24T09:55:00+01:00",
 2.0.7     "mud-signature": "https://our-factory.com/mud/general-signature.p7s",
 2.0.8     "cache-validity": 40,
 2.0.9     "is-supported": true,
2.0.10     "systeminfo": "Industrial Edge AI Gateway (BTC22)",
2.0.11     "from-device-policy": {
2.0.12       "access-lists": {
2.0.13         "access-list": [
2.0.14           {
2.0.15             "name": "from-ipv4-edge-gateway"
2.0.16           }
2.0.17         ]
2.0.18       }
2.0.19     },
2.0.20     "to-device-policy": {
2.0.21       "access-lists": {
2.0.22         "access-list": [
2.0.23           {
2.0.24             "name": "to-ipv4-edge-gateway"
2.0.25           }
2.0.26         ]
2.0.27       }
2.0.28     }
2.0.29   },
2.0.30   "ietf-access-control-list:acls": {
```

```
2.0.31    "acl": [
2.0.32      {
2.0.33        "name": "from-ipv4-edge-gateway",
2.0.34        "type": "ietf-access-control-list:ipv4-acl-type",
2.0.35        "aces": {
2.0.36          "ace": [
2.0.37            {
2.0.38              "name": "from-cloud-telemetry-https",
2.0.39              "matches": {
2.0.40                "ipv4": {
2.0.41                  "protocol": 6,
2.0.42                  "ietf-acldns:dst-dnsname": "aws.amazon.com"
2.0.43                },
2.0.44                "tcp": {
2.0.45                  "destination-port": {
2.0.46                    "operator": "eq",
2.0.47                    "port": 443
2.0.48                  },
2.0.49                  "ietf-mud:direction-initiated": "from-device"
2.0.50                }
2.0.51              },
2.0.52              "actions": {
2.0.53                "forwarding": "accept"
2.0.54              }
2.0.55            },
2.0.56            {
2.0.57              "name": "from-cloud-telemetry-mqtts",
2.0.58              "matches": {
2.0.59                "ipv4": {
2.0.60                  "protocol": 6,
2.0.61                  "ietf-acldns:dst-dnsname": "aws.amazon.com"
2.0.62                },
2.0.63                "tcp": {
2.0.64                  "destination-port": {
2.0.65                    "operator": "eq",
2.0.66                    "port": 8883
2.0.67                  },
2.0.68                  "ietf-mud:direction-initiated": "from-device"
2.0.69                }
2.0.70              },
2.0.71              "actions": {
```

```
2.0.72                "forwarding": "accept"
2.0.73              }
2.0.74            },
2.0.75            {
2.0.76              "name": "from-safety-halt",
2.0.77              "matches": {
2.0.78                "ietf-mud:mud": {
2.0.79                  "controller": "urn:our-factory:controller:safety-relay"
2.0.80                },
2.0.81                "ipv4": {
2.0.82                  "protocol": 6
2.0.83                },
2.0.84                "tcp": {
2.0.85                  "destination-port": {
2.0.86                    "operator": "eq",
2.0.87                    "port": 502
2.0.88                  },
2.0.89                  "ietf-mud:direction-initiated": "from-device"
2.0.90                }
2.0.91              },
2.0.92              "actions": {
2.0.93                "forwarding": "accept"
2.0.94              }
2.0.95            },
2.0.96            {
2.0.97              "name": "from-dns-query",
2.0.98              "matches": {
2.0.99                "ietf-mud:mud": {
2.0.100                 "controller": "urn:ietf:params:mud:dns"
2.0.101               },
2.0.102               "ipv4": {
2.0.103                 "protocol": 17
2.0.104               },
2.0.105               "udp": {
2.0.106                 "destination-port": {
2.0.107                   "operator": "eq",
2.0.108                   "port": 53
2.0.109                 }
2.0.110               }
2.0.111             },
2.0.112             "actions": {
```

```
2.0.113                 "forwarding": "accept"
2.0.114               }
2.0.115             },
2.0.116             {
2.0.117               "name": "from-ntp-sync",
2.0.118               "matches": {
2.0.119                 "ietf-mud:mud": {
2.0.120                   "controller": "urn:ietf:params:mud:ntp"
2.0.121                 },
2.0.122                 "ipv4": {
2.0.123                   "protocol": 17
2.0.124                 },
2.0.125                 "udp": {
2.0.126                   "destination-port": {
2.0.127                     "operator": "eq",
2.0.128                     "port": 123
2.0.129                   }
2.0.130                 }
2.0.131               },
2.0.132               "actions": {
2.0.133                 "forwarding": "accept"
2.0.134               }
2.0.135             },
2.0.136             {
2.0.137               "name": "from-mqtt-inbound-reply",
2.0.138               "matches": {
2.0.139                 "ietf-mud:mud": {
2.0.140                   "controller": "urn:our-factory:controller:mqtt-bridge"
2.0.141                 },
2.0.142                 "ipv4": {
2.0.143                   "protocol": 6
2.0.144                 },
2.0.145                 "tcp": {
2.0.146                   "source-port": {
2.0.147                     "operator": "eq",
2.0.148                     "port": 1883
2.0.149                   }
2.0.150                 }
2.0.151               },
2.0.152               "actions": {
2.0.153                 "forwarding": "accept"
```

```
2.0.154               }
2.0.155             },
2.0.156             {
2.0.157               "name": "from-coap-audio-inbound-reply",
2.0.158               "matches": {
2.0.159                 "ietf-mud:mud": {
2.0.160                   "controller": "urn:our-factory:controller:acoustic-
          sensor"
2.0.161                 },
2.0.162                 "ipv4": {
2.0.163                   "protocol": 17
2.0.164                 },
2.0.165                 "udp": {
2.0.166                   "source-port": {
2.0.167                     "operator": "eq",
2.0.168                     "port": 5683
2.0.169                   }
2.0.170                 }
2.0.171               },
2.0.172               "actions": {
2.0.173                 "forwarding": "accept"
2.0.174               }
2.0.175             }
2.0.176           ]
2.0.177         }
2.0.178       },
2.0.179       {
2.0.180         "name": "to-ipv4-edge-gateway",
2.0.181         "type": "ietf-access-control-list:ipv4-acl-type",
2.0.182         "aces": {
2.0.183           "ace": [
2.0.184             {
2.0.185               "name": "to-mqtt-inbound",
2.0.186               "matches": {
2.0.187                 "ietf-mud:mud": {
2.0.188                   "controller": "urn:our-factory:controller:mqtt-bridge"
2.0.189                 },
2.0.190                 "ipv4": {
2.0.191                   "protocol": 6
2.0.192                 },
2.0.193                 "tcp": {
```

```
2.0.194                 "destination-port": {
2.0.195                   "operator": "eq",
2.0.196                   "port": 1883
2.0.197                 },
2.0.198                 "ietf-mud:direction-initiated": "to-device"
2.0.199               }
2.0.200             },
2.0.201             "actions": {
2.0.202               "forwarding": "accept"
2.0.203             }
2.0.204           },
2.0.205           {
2.0.206             "name": "to-coap-audio-inbound",
2.0.207             "matches": {
2.0.208               "ietf-mud:mud": {
2.0.209                 "controller": "urn:our-factory:controller:acoustic-
        sensor"
2.0.210               },
2.0.211               "ipv4": {
2.0.212                 "protocol": 17
2.0.213               },
2.0.214               "udp": {
2.0.215                 "destination-port": {
2.0.216                   "operator": "eq",
2.0.217                   "port": 5683
2.0.218                 }
2.0.219               }
2.0.220             },
2.0.221             "actions": {
2.0.222               "forwarding": "accept"
2.0.223             }
2.0.224           },
2.0.225           {
2.0.226             "name": "to-cloud-telemetry-https-reply",
2.0.227             "matches": {
2.0.228               "ipv4": {
2.0.229                 "protocol": 6,
2.0.230                 "ietf-acldns:src-dnsname": "aws.amazon.com"
2.0.231               },
2.0.232               "tcp": {
2.0.233                 "source-port": {
```

```
2.0.234                    "operator": "eq",
2.0.235                    "port": 443
2.0.236                  }
2.0.237                }
2.0.238              },
2.0.239              "actions": {
2.0.240                "forwarding": "accept"
2.0.241              }
2.0.242            },
2.0.243            {
2.0.244              "name": "to-cloud-telemetry-mqtts-reply",
2.0.245              "matches": {
2.0.246                "ipv4": {
2.0.247                  "protocol": 6,
2.0.248                  "ietf-acldns:src-dnsname": "aws.amazon.com"
2.0.249                },
2.0.250                "tcp": {
2.0.251                  "source-port": {
2.0.252                    "operator": "eq",
2.0.253                    "port": 8883
2.0.254                  }
2.0.255                }
2.0.256              },
2.0.257              "actions": {
2.0.258                "forwarding": "accept"
2.0.259              }
2.0.260            },
2.0.261            {
2.0.262              "name": "to-safety-halt-reply",
2.0.263              "matches": {
2.0.264                "ietf-mud:mud": {
2.0.265                  "controller": "urn:our-factory:controller:safety-relay"
2.0.266                },
2.0.267                "ipv4": {
2.0.268                  "protocol": 6
2.0.269                },
2.0.270                "tcp": {
2.0.271                  "source-port": {
2.0.272                    "operator": "eq",
2.0.273                    "port": 502
2.0.274                  }
```

```
2.0.275                 }
2.0.276               },
2.0.277               "actions": {
2.0.278                 "forwarding": "accept"
2.0.279               }
2.0.280             },
2.0.281             {
2.0.282               "name": "to-dns-query-reply",
2.0.283               "matches": {
2.0.284                 "ietf-mud:mud": {
2.0.285                   "controller": "urn:ietf:params:mud:dns"
2.0.286                 },
2.0.287                 "ipv4": {
2.0.288                   "protocol": 17
2.0.289                 },
2.0.290                 "udp": {
2.0.291                   "source-port": {
2.0.292                     "operator": "eq",
2.0.293                     "port": 53
2.0.294                   }
2.0.295                 }
2.0.296               },
2.0.297               "actions": {
2.0.298                 "forwarding": "accept"
2.0.299               }
2.0.300             },
2.0.301             {
2.0.302               "name": "to-ntp-sync-reply",
2.0.303               "matches": {
2.0.304                 "ietf-mud:mud": {
2.0.305                   "controller": "urn:ietf:params:mud:ntp"
2.0.306                 },
2.0.307                 "ipv4": {
2.0.308                   "protocol": 17
2.0.309                 },
2.0.310                 "udp": {
2.0.311                   "source-port": {
2.0.312                     "operator": "eq",
2.0.313                     "port": 123
2.0.314                   }
2.0.315                 }
```

```
2.0.316               },
2.0.317               "actions": {
2.0.318                 "forwarding": "accept"
2.0.319               }
2.0.320             }
2.0.321           ]
2.0.322         }
2.0.323       }
2.0.324     ]
2.0.325   }
2.0.326 }
2.0.327 }
```

B.1.5 Networked safety block

```
 2.0.1 {
 2.0.2   {
 2.0.3   "ietf-mud:mud": {
 2.0.4     "mud-version": 1,
 2.0.5     "mud-url": "https://our-factory.com/mud/safety-block.json",
 2.0.6     "last-update": "2026-03-22T12:05:00+01:00",
 2.0.7     "mud-signature": "https://our-factory.com/mud/general-signature.p7s",
 2.0.8     "cache-validity": 40,
 2.0.9     "is-supported": true,
2.0.10     "systeminfo": "Networked Safety Contactor / Safety Block",
2.0.11     "from-device-policy": {
2.0.12       "access-lists": {
2.0.13         "access-list": [
2.0.14           {
2.0.15             "name": "from-ipv4-safety-block"
2.0.16           }
2.0.17         ]
2.0.18       }
2.0.19     },
2.0.20     "to-device-policy": {
2.0.21       "access-lists": {
2.0.22         "access-list": [
2.0.23           {
2.0.24             "name": "to-ipv4-safety-block"
2.0.25           }
```

```
2.0.26          ]
2.0.27        }
2.0.28      }
2.0.29    },
2.0.30    "ietf-access-control-list:acls": {
2.0.31      "acl": [
2.0.32        {
2.0.33          "name": "from-ipv4-safety-block",
2.0.34          "type": "ipv4-acl-type",
2.0.35          "aces": {
2.0.36            "ace": [
2.0.37              {
2.0.38                "name": "from-modbus-halt-command-reply",
2.0.39                "matches": {
2.0.40                  "ietf-mud:mud": {
2.0.41                    "my-controller": [null]
2.0.42                  },
2.0.43                  "ipv4": {
2.0.44                    "protocol": 6
2.0.45                  },
2.0.46                  "tcp": {
2.0.47                    "source-port": {
2.0.48                      "operator": "eq",
2.0.49                      "port": 502
2.0.50                    }
2.0.51                  }
2.0.52                },
2.0.53                "actions": {
2.0.54                  "forwarding": "accept"
2.0.55                }
2.0.56              }
2.0.57            ]
2.0.58          }
2.0.59        },
2.0.60        {
2.0.61          "name": "to-ipv4-safety-block",
2.0.62          "type": "ipv4-acl-type",
2.0.63          "aces": {
2.0.64            "ace": [
2.0.65              {
2.0.66                "name": "to-modbus-halt-command",
```

```
2.0.67                "matches": {
2.0.68                  "ietf-mud:mud": {
2.0.69                    "my-controller": [null]
2.0.70                  },
2.0.71                  "ipv4": {
2.0.72                    "protocol": 6
2.0.73                  },
2.0.74                  "tcp": {
2.0.75                    "destination-port": {
2.0.76                      "operator": "eq",
2.0.77                      "port": 502
2.0.78                    },
2.0.79                    "ietf-mud:direction-initiated": "to-device"
2.0.80                  }
2.0.81                },
2.0.82                "actions": {
2.0.83                  "forwarding": "accept"
2.0.84                }
2.0.85              }
2.0.86            ]
2.0.87          }
2.0.88        }
2.0.89      ]
2.0.90    }
2.0.91 }
2.0.92 }
```

Appendix C
Scenario registry


```
3.0.1 {
3.0.2   {
3.0.3   "registry_version": "1.0.0",
3.0.4   "scenarios": [
3.0.5     {
3.0.6       "scenario_id": "B1",
3.0.7       "variant_group_id": "B1",
3.0.8       "dataset_kind": "baseline",
3.0.9       "stage": "baseline",
3.0.10      "file_name": "01_acoustic_sensor.json",
3.0.11      "relative_file_path": "mud_files/baseline_scenario/01
          _acoustic_sensor.json",
3.0.12      "device_under_test": "Acoustic Sensor",
3.0.13      "scenario_title": "Baseline Acoustic Sensor",
3.0.14      "manipulation_description": "This scenario does not involve any
          manipulation. It utilizes the baseline reference Manufacturer Usage
          Description (MUD) file of the acoustic sensor for baseline comparison
          purposes.",
3.0.15      "expected_conformance_level": "Platinum",
3.0.16      "expected_functional_security_level": "Secure",
3.0.17      "expected_accuracy_class": "accurate",
3.0.18      "notes": "Baseline scenario file.",
3.0.19      "aliases": []
3.0.20    },
3.0.21    {
3.0.22      "scenario_id": "B2",
3.0.23      "variant_group_id": "B2",
3.0.24      "dataset_kind": "baseline",
3.0.25      "stage": "baseline",
3.0.26      "file_name": "04_edge_gateway.json",
3.0.27      "relative_file_path": "mud_files/baseline_scenario/04_edge_gateway.
          json",
3.0.28      "device_under_test": "Edge Gateway",
3.0.29      "scenario_title": "Baseline Edge Gateway",
```

```
3.0.30      "manipulation_description": "This scenario does not involve any
        manipulation. It utilizes the baseline reference Manufacturer Usage
        Description (MUD) file of the edge gateway for baseline comparison
        purposes.",
3.0.31      "expected_conformance_level": "Platinum",
3.0.32      "expected_functional_security_level": "Secure",
3.0.33      "expected_accuracy_class": "accurate",
3.0.34      "notes": "Baseline scenario file.",
3.0.35      "aliases": []
3.0.36    },
3.0.37    {
3.0.38      "scenario_id": "B3",
3.0.39      "variant_group_id": "B3",
3.0.40      "dataset_kind": "baseline",
3.0.41      "stage": "baseline",
3.0.42      "file_name": "03_mqtt_bridge.json",
3.0.43      "relative_file_path": "mud_files/baseline_scenario/03_mqtt_bridge.
        json",
3.0.44      "device_under_test": "mqtt-bridge",
3.0.45      "scenario_title": "Baseline mqtt-bridge",
3.0.46      "manipulation_description": "This scenario does not involve any
        manipulation. It utilizes the baseline reference Manufacturer Usage
        Description (MUD) file of the MQTT bridge for baseline comparison
        purposes.",
3.0.47      "expected_conformance_level": "Platinum",
3.0.48      "expected_functional_security_level": "Secure",
3.0.49      "expected_accuracy_class": "accurate",
3.0.50      "notes": "Baseline scenario file.",
3.0.51      "aliases": []
3.0.52    },
3.0.53    {
3.0.54      "scenario_id": "B4",
3.0.55      "variant_group_id": "B4",
3.0.56      "dataset_kind": "baseline",
3.0.57      "stage": "baseline",
3.0.58      "file_name": "05_safety_block.json",
3.0.59      "relative_file_path": "mud_files/baseline_scenario/05_safety_block.
        json",
3.0.60      "device_under_test": "Safety Block",
3.0.61      "scenario_title": "Baseline Safety Block",
```

```
    "manipulation_description": "This scenario does not involve any manipulation. It utilizes the baseline reference Manufacturer Usage Description (MUD) file of the safety block for baseline comparison purposes.",
    "expected_conformance_level": "Platinum",
    "expected_functional_security_level": "Secure",
    "expected_accuracy_class": "accurate",
    "notes": "Baseline scenario file.",
    "aliases": []
  },
  {
    "scenario_id": "B5",
    "variant_group_id": "B5",
    "dataset_kind": "baseline",
    "stage": "baseline",
    "file_name": "02_vibration_sensor.json",
    "relative_file_path": "mud_files/baseline_scenario/02_vibration_sensor.json",
    "device_under_test": "Vibration Sensor",
    "scenario_title": "Baseline Vibration Sensor",
    "manipulation_description": "This scenario does not involve any manipulation. It utilizes the baseline reference Manufacturer Usage Description (MUD) file of the vibration sensor for baseline comparison purposes.",
    "expected_conformance_level": "Platinum",
    "expected_functional_security_level": "Secure",
    "expected_accuracy_class": "accurate",
    "notes": "Baseline scenario file.",
    "aliases": []
  },
  {
    "scenario_id": "B1_NTN",
    "variant_group_id": "B1",
    "dataset_kind": "manipulated_w_o_telling_names",
    "stage": "baseline",
    "file_name": "1001_acoustic_sensor.json",
    "relative_file_path": "mud_files/manipulated_w_o_telling_names/1001_acoustic_sensor.json",
    "device_under_test": "Acoustic Sensor",
    "scenario_title": "Baseline Acoustic Sensor (No System Info)",
```

```
    "manipulation_description": "This is a baseline configuration of the acoustic sensor modified for context-ablation analysis, wherein the `system-info` field has been excluded, creating a 'no system info' (NTN) variant.",
    "expected_conformance_level": "Gold",
    "expected_functional_security_level": "Secure",
    "expected_accuracy_class": "accurate",
    "notes": "No-system-info variant of B1 baseline.",
    "aliases": []
  },
  {
    "scenario_id": "B2_NTN",
    "variant_group_id": "B2",
    "dataset_kind": "manipulated_w_o_telling_names",
    "stage": "baseline",
    "file_name": "1002_vibration_sensor.json",
    "relative_file_path": "mud_files/manipulated_w_o_telling_names/1002_vibration_sensor.json",
    "device_under_test": "Vibration Sensor",
    "scenario_title": "Baseline Vibration Sensor (No System Info)",
    "manipulation_description": "This is a baseline configuration of the vibration sensor modified for context-ablation analysis, wherein the `system-info` field has been excluded, creating a 'no system info' (NTN) variant.",
    "expected_conformance_level": "Gold",
    "expected_functional_security_level": "Secure",
    "expected_accuracy_class": "accurate",
    "notes": "No-system-info variant of B2 baseline.",
    "aliases": []
  },
  {
    "scenario_id": "B3_NTN",
    "variant_group_id": "B3",
    "dataset_kind": "manipulated_w_o_telling_names",
    "stage": "baseline",
    "file_name": "1003_mqtt_bridge.json",
    "relative_file_path": "mud_files/manipulated_w_o_telling_names/1003_mqtt_bridge.json",
    "device_under_test": "mqtt-bridge",
    "scenario_title": "Baseline mqtt-bridge (No System Info)",
```

```
3.0.126       "manipulation_description": "This is a baseline configuration of the
          MQTT bridge modified for context-ablation analysis, wherein the `
         system-info` field has been excluded, creating a 'no system info' (NTN
         ) variant.",
3.0.127       "expected_conformance_level": "Gold",
3.0.128       "expected_functional_security_level": "Secure",
3.0.129       "expected_accuracy_class": "accurate",
3.0.130       "notes": "No-system-info variant of B3 baseline.",
3.0.131       "aliases": []
3.0.132     },
3.0.133     {
3.0.134       "scenario_id": "B4_NTN",
3.0.135       "variant_group_id": "B4",
3.0.136       "dataset_kind": "manipulated_w_o_telling_names",
3.0.137       "stage": "baseline",
3.0.138       "file_name": "1004_edge_gateway.json",
3.0.139       "relative_file_path": "mud_files/manipulated_w_o_telling_names/1004
         _edge_gateway.json",
3.0.140       "device_under_test": "Edge Gateway",
3.0.141       "scenario_title": "Baseline Edge Gateway (No System Info)",
3.0.142       "manipulation_description": "This is a baseline configuration of the
          edge gateway modified for context-ablation analysis, wherein the `
         system-info` field has been excluded, creating a 'no system info' (NTN
         ) variant.",
3.0.143       "expected_conformance_level": "Gold",
3.0.144       "expected_functional_security_level": "Secure",
3.0.145       "expected_accuracy_class": "accurate",
3.0.146       "notes": "No-system-info variant of B4 baseline.",
3.0.147       "aliases": []
3.0.148     },
3.0.149     {
3.0.150       "scenario_id": "B5_NTN",
3.0.151       "variant_group_id": "B5",
3.0.152       "dataset_kind": "manipulated_w_o_telling_names",
3.0.153       "stage": "baseline",
3.0.154       "file_name": "1005_safety_block.json",
3.0.155       "relative_file_path": "mud_files/manipulated_w_o_telling_names/1005
         _safety_block.json",
3.0.156       "device_under_test": "Safety Block",
3.0.157       "scenario_title": "Baseline Safety Block (No System Info)",
```

```
    "manipulation_description": "This is a baseline configuration of the safety block modified for context-ablation analysis, wherein the `system-info` field has been excluded, creating a 'no system info' (NTN) variant.",
    "expected_conformance_level": "Gold",
    "expected_functional_security_level": "Secure",
    "expected_accuracy_class": "accurate",
    "notes": "No-system-info variant of B5 baseline.",
    "aliases": []
  },
  {
    "scenario_id": "S1",
    "variant_group_id": "S1",
    "dataset_kind": "manipulated",
    "stage": "syntax_error",
    "file_name": "101_missing_mud_container.json",
    "relative_file_path": "mud_files/manipulated/101_missing_mud_container.json",
    "device_under_test": "Vibration Sensor",
    "scenario_title": "Missing MUD Container",
    "manipulation_description": "The standard `ietf-mud:mud` container has been intentionally omitted from the vibration sensor's MUD file to test the system's resilience against fundamental structural malformations.",
    "expected_conformance_level": "Failed",
    "expected_functional_security_level": "N/A",
    "expected_accuracy_class": "none",
    "notes": "Should fail parsing or YANG validation early.",
    "aliases": []
  },
  {
    "scenario_id": "S2",
    "variant_group_id": "S2",
    "dataset_kind": "manipulated",
    "stage": "syntax_error",
    "file_name": "102_typo_in_acoustic_sensor.json",
    "relative_file_path": "mud_files/manipulated/102_typo_in_acoustic_sensor.json",
    "device_under_test": "Acoustic Sensor",
    "scenario_title": "Typo in Policy Field",
```

```
    "manipulation_description": "A deliberate typographical error has been introduced into the acoustic sensor's policy field, specifically substituting `to-device-policy` with `to-devce-polcy`, to evaluate syntactic validation capabilities.",
    "expected_conformance_level": "Failed",
    "expected_functional_security_level": "N/A",
    "expected_accuracy_class": "none",
    "notes": "Expected to fail YANG validation.",
    "aliases": []
  },
  {
    "scenario_id": "S3",
    "variant_group_id": "S3",
    "dataset_kind": "manipulated",
    "stage": "syntax_error",
    "file_name": "103_not_register_acls.json",
    "relative_file_path": "mud_files/manipulated/103_not_register_acls.json",
    "device_under_test": "Edge Gateway",
    "scenario_title": "Missing ACL Registration",
    "manipulation_description": "The access control list registrations have been purposely excluded from the top-level MUD container of the edge gateway to assess the system's capability in identifying missing fundamental components.",
    "expected_conformance_level": "Failed",
    "expected_functional_security_level": "N/A",
    "expected_accuracy_class": "none",
    "notes": "Expected to fail YANG validation.",
    "aliases": []
  },
  {
    "scenario_id": "S4",
    "variant_group_id": "S4",
    "dataset_kind": "manipulated",
    "stage": "syntax_error",
    "file_name": "104_missing_comma.json",
    "relative_file_path": "mud_files/manipulated/104_missing_comma.json",
    "device_under_test": "Safety Block",
    "scenario_title": "Malformed JSON Comma Removal",
```

```
3.0.222     "manipulation_description": "A critical comma separator has been
        deliberately deleted between the `ietf-mud:mud` and `ietf-mud:access-
        lists` containers within the safety block's file to simulate a
        malformed JSON object.",
3.0.223     "expected_conformance_level": "Failed",
3.0.224     "expected_functional_security_level": "N/A",
3.0.225     "expected_accuracy_class": "none",
3.0.226     "notes": "Expected to fail JSON parsing.",
3.0.227     "aliases": []
3.0.228   },
3.0.229   {
3.0.230     "scenario_id": "S5",
3.0.231     "variant_group_id": "S5",
3.0.232     "dataset_kind": "manipulated",
3.0.233     "stage": "conformance_testing",
3.0.234     "file_name": "105_mandatory_fields_only.json",
3.0.235     "relative_file_path": "mud_files/manipulated/105
        _mandatory_fields_only.json",
3.0.236     "device_under_test": "mqtt-bridge",
3.0.237     "scenario_title": "Mandatory Fields Only",
3.0.238     "manipulation_description": "The MUD file of the MQTT bridge has
        been truncated to retain exclusively mandatory fields, thereby
        facilitating an evaluation of conformance-level reduction without
        introducing functional security risks.",
3.0.239     "expected_conformance_level": "Silver",
3.0.240     "expected_functional_security_level": "Secure",
3.0.241     "expected_accuracy_class": "accurate",
3.0.242     "notes": "Conformance reduction without security risk.",
3.0.243     "aliases": []
3.0.244   },
3.0.245   {
3.0.246     "scenario_id": "S6",
3.0.247     "variant_group_id": "S6",
3.0.248     "dataset_kind": "manipulated",
3.0.249     "stage": "conformance_testing",
3.0.250     "file_name": "106_mandatory_and_signed.json",
3.0.251     "relative_file_path": "mud_files/manipulated/106
        _mandatory_and_signed.json",
3.0.252     "device_under_test": "Edge Gateway",
3.0.253     "scenario_title": "Mandatory and Signed",
```

```
3.0.254     "manipulation_description": "The edge gateway's MUD file maintains
        mandatory fields accompanied by a valid signature, whilst omitting
        subsequent optional requirements, to verify exact conformance tier
        classification.",
3.0.255     "expected_conformance_level": "Gold",
3.0.256     "expected_functional_security_level": "Secure",
3.0.257     "expected_accuracy_class": "accurate",
3.0.258     "notes": "Conformance level expected to be Gold.",
3.0.259     "aliases": []
3.0.260   },
3.0.261   {
3.0.262     "scenario_id": "S7",
3.0.263     "variant_group_id": "S7",
3.0.264     "dataset_kind": "manipulated",
3.0.265     "stage": "conformance_testing",
3.0.266     "file_name": "107_perfect_comformance.json",
3.0.267     "relative_file_path": "mud_files/manipulated/107_perfect_comformance
        .json",
3.0.268     "device_under_test": "Vibration Sensor",
3.0.269     "scenario_title": "Perfect Conformance",
3.0.270     "manipulation_description": "A fully compliant reference MUD file is
         provided for the vibration sensor to act as a definitive standard
        benchmark for perfect conformance tracking.",
3.0.271     "expected_conformance_level": "Platinum",
3.0.272     "expected_functional_security_level": "Secure",
3.0.273     "expected_accuracy_class": "accurate",
3.0.274     "notes": "Reference compliant file.",
3.0.275     "aliases": []
3.0.276   },
3.0.277   {
3.0.278     "scenario_id": "S17",
3.0.279     "variant_group_id": "S17",
3.0.280     "dataset_kind": "manipulated",
3.0.281     "stage": "conformance_testing",
3.0.282     "file_name": "117_bronze_conformance.json",
3.0.283     "relative_file_path": "mud_files/manipulated/117_bronze_conformance.
        json",
3.0.284     "device_under_test": "Edge Gateway",
3.0.285     "scenario_title": "Bronze Level Conformance",
```

```
3.0.286     "manipulation_description": "The edge gateway's MUD file is
        structured to reflect a Bronze level conformance standard,
        establishing a lower-tier compliance benchmark for evaluation.",
3.0.287     "expected_conformance_level": "Bronze",
3.0.288     "expected_functional_security_level": "Secure",
3.0.289     "expected_accuracy_class": "accurate",
3.0.290     "notes": "Conformance level expected to be Bronze.",
3.0.291     "aliases": []
3.0.292   },
3.0.293   {
3.0.294     "scenario_id": "S8",
3.0.295     "variant_group_id": "S8",
3.0.296     "dataset_kind": "manipulated",
3.0.297     "stage": "v_library_testing",
3.0.298     "file_name": "108_http_downgrade.json",
3.0.299     "relative_file_path": "mud_files/manipulated/108_http_downgrade.json
        ",
3.0.300     "device_under_test": "Edge Gateway",
3.0.301     "scenario_title": "HTTP Downgrade",
3.0.302     "manipulation_description": "An explicit rule downgrade has been
        introduced to the edge gateway, transitioning secure HTTPS port 443
        traffic to insecure HTTP port 80 traffic, in order to test the
        detection of exposed insecure protocols.",
3.0.303     "expected_conformance_level": "Platinum",
3.0.304     "expected_functional_security_level": "Insecure",
3.0.305     "expected_accuracy_class": "over_permissive",
3.0.306     "notes": "V-library scenario (insecure service/protocol exposure).",
3.0.307     "aliases": []
3.0.308   },
3.0.309   {
3.0.310     "scenario_id": "S9",
3.0.311     "variant_group_id": "S9",
3.0.312     "dataset_kind": "manipulated",
3.0.313     "stage": "v_library_testing",
3.0.314     "file_name": "109_left_ssh_port.json",
3.0.315     "relative_file_path": "mud_files/manipulated/109_left_ssh_port.json"
        ,
3.0.316     "device_under_test": "Edge Gateway",
3.0.317     "scenario_title": "Leftover SSH Maintenance Port",
3.0.318     "manipulation_description": "An extraneous inbound TCP rule
        permitting traffic on port 22 from local networks has been injected
```

```
        into the edge gateway configuration to simulate an inadvertently
        exposed maintenance SSH port.",
3.0.319   "expected_conformance_level": "Platinum",
3.0.320   "expected_functional_security_level": "Insecure",
3.0.321   "expected_accuracy_class": "over_permissive",
3.0.322   "notes": "V-library vulnerability scenario (exposed maintenance port
        ).",
3.0.323   "aliases": []
3.0.324  },
3.0.325  {
3.0.326   "scenario_id": "S10",
3.0.327   "variant_group_id": "S10",
3.0.328   "dataset_kind": "manipulated",
3.0.329   "stage": "accuracy_testing",
3.0.330   "file_name": "110_missing_internet.json",
3.0.331   "relative_file_path": "mud_files/manipulated/110_missing_internet.
        json",
3.0.332   "device_under_test": "Edge Gateway",
3.0.333   "scenario_title": "Missing Internet Connectivity",
3.0.334   "manipulation_description": "The requisite rules permitting internet
         connectivity for the edge gateway have been eliminated to model an
        excessively restrictive, under-permissive configuration anomaly.",
3.0.335   "expected_conformance_level": "Platinum",
3.0.336   "expected_functional_security_level": "Insecure",
3.0.337   "expected_accuracy_class": "under_permissive",
3.0.338   "notes": "Under-permissive scenario.",
3.0.339   "aliases": []
3.0.340  },
3.0.341  {
3.0.342   "scenario_id": "S12",
3.0.343   "variant_group_id": "S12",
3.0.344   "dataset_kind": "manipulated",
3.0.345   "stage": "accuracy_testing",
3.0.346   "file_name": "112_missing_dns.json",
3.0.347   "relative_file_path": "mud_files/manipulated/112_missing_dns.json",
3.0.348   "device_under_test": "Edge Gateway",
3.0.349   "scenario_title": "Missing DNS Connectivity",
3.0.350   "manipulation_description": "The Domain Name System (DNS) query
        access rules have been eradicated from the edge gateway, impeding
        cloud telemetry hostname resolution and testing the detection of
        missing indispensable services.",
```

```
    "expected_conformance_level": "Platinum",
    "expected_functional_security_level": "Secure",
    "expected_accuracy_class": "under_permissive",
    "notes": "will it detect the missing DNS ACEs?",
    "aliases": []
  },
  {
    "scenario_id": "S11",
    "variant_group_id": "S11",
    "dataset_kind": "manipulated",
    "stage": "accuracy_testing",
    "file_name": "111_half_a_standard.json",
    "relative_file_path": "mud_files/manipulated/111_half_a_standard.
json",
    "device_under_test": "Vibration Sensor",
    "scenario_title": "Missing Half of a Standard",
    "manipulation_description": "The bidirectional UDP port 320
Precision Time Protocol (PTP) general rule has been discarded while
retaining the UDP port 319 event rule, modeling an incomplete standard
 implementation for the vibration sensor.",
    "expected_conformance_level": "Platinum",
    "expected_functional_security_level": "Insecure",
    "expected_accuracy_class": "under_permissive",
    "notes": "Under-permissive scenario.",
    "aliases": []
  },
  {
    "scenario_id": "S13",
    "variant_group_id": "S13",
    "dataset_kind": "manipulated",
    "stage": "accuracy_testing",
    "file_name": "113_open_door.json",
    "relative_file_path": "mud_files/manipulated/113_open_door.json",
    "device_under_test": "Edge Gateway",
    "scenario_title": "Open Door",
    "manipulation_description": "An unsolicited inbound TCP port 443
rule devoid of specific address constraints has been appended to the
edge gateway, reflecting an over-permissive and broadly exposed
network posture.",
    "expected_conformance_level": "Platinum",
    "expected_functional_security_level": "Insecure",
```

```
    "expected_accuracy_class": "over_permissive",
    "notes": "Over-permissive scenario.",
    "aliases": []
  },
  {
    "scenario_id": "S14",
    "variant_group_id": "S14",
    "dataset_kind": "manipulated",
    "stage": "accuracy_testing",
    "file_name": "114_silent_data_exfil.json",
    "relative_file_path": "mud_files/manipulated/114_silent_data_exfil.
json",
    "device_under_test": "mqtt-bridge",
    "scenario_title": "Silent Data Exfiltration",
    "manipulation_description": "An insidious outbound exfiltration
channel leveraging UDP, DNS, or ICMP via internet egress has been
established in the MQTT bridge to evaluate the identification of
silent data leakage vectors.",
    "expected_conformance_level": "Platinum",
    "expected_functional_security_level": "Insecure",
    "expected_accuracy_class": "over_permissive",
    "notes": ,
    "aliases": [
      "silent_data_exfiltration"
    ]
  },
  {
    "scenario_id": "S15",
    "variant_group_id": "S15",
    "dataset_kind": "manipulated",
    "stage": "accuracy_testing",
    "file_name": "115_implausible_connection.json",
    "relative_file_path": "mud_files/manipulated/115
_implausible_connection.json",
    "device_under_test": "Safety Block",
    "scenario_title": "Implausible Internet Connection",
    "manipulation_description": "An exceptionally implausible inbound
internet reachability rule originating from a public IP address has
been assigned to the mission-critical safety block device to test
anomaly-based intrusion rules.",
    "expected_conformance_level": "Platinum",
```

```
    "expected_functional_security_level": "Insecure",
    "expected_accuracy_class": "over_permissive",
    "notes": null,
    "aliases": []
  },
  {
    "scenario_id": "S16",
    "variant_group_id": "S16",
    "dataset_kind": "manipulated",
    "stage": "accuracy_testing",
    "file_name": "116_lateral_movement.json",
    "relative_file_path": "mud_files/manipulated/116_lateral_movement.
json",
    "device_under_test": "Edge Gateway",
    "scenario_title": "Lateral Movement Pivot",
    "manipulation_description": "An anomalous initiated TCP port 445 (
SMB) rule targeting local networks has been injected into the edge
gateway to simulate potential pivot behaviors associated with lateral
movement attacks.",
    "expected_conformance_level": "Platinum",
    "expected_functional_security_level": "Insecure",
    "expected_accuracy_class": "over_permissive",
    "notes": "Over-permissive scenario.",
    "aliases": []
  },
  {
    "scenario_id": "S8_NTN",
    "variant_group_id": "S8",
    "dataset_kind": "manipulated_w_o_telling_names",
    "stage": "v_library_testing",
    "file_name": "1008_http_downgrade.json",
    "relative_file_path": "mud_files/manipulated_w_o_telling_names/1008
_http_downgrade.json",
    "device_under_test": "Edge Gateway",
    "scenario_title": "HTTP Downgrade (No System-Info Variant)",
    "manipulation_description": "An explicit rule downgrade has been
introduced, transitioning secure HTTPS traffic to insecure HTTP
traffic, while the `system-info` metadata has been excluded to serve
as a 'no system info' (NTN) variant.",
    "expected_conformance_level": "Gold",
    "expected_functional_security_level": "Insecure",
```

```
    "expected_accuracy_class": "over_permissive",
    "notes": "No-system-info variant of S8.",
    "aliases": []
  },
  {
    "scenario_id": "S9_NTN",
    "variant_group_id": "S9",
    "dataset_kind": "manipulated_w_o_telling_names",
    "stage": "v_library_testing",
    "file_name": "1009_left_ssh_port.json",
    "relative_file_path": "mud_files/manipulated_w_o_telling_names/1009_left_ssh_port.json",
    "device_under_test": "Edge Gateway",
    "scenario_title": "Leftover SSH Maintenance Port (No System-Info Variant)",
    "manipulation_description": "An extraneous inbound TCP rule for SSH on port 22 has been injected to simulate an exposed maintenance port, simultaneously lacking the `system-info` attribute as a 'no system info' (NTN) variant.",
    "expected_conformance_level": "Gold",
    "expected_functional_security_level": "Insecure",
    "expected_accuracy_class": "over_permissive",
    "notes": "No-system-info variant of S9.",
    "aliases": []
  },
  {
    "scenario_id": "S10_NTN",
    "variant_group_id": "S10",
    "dataset_kind": "manipulated_w_o_telling_names",
    "stage": "accuracy_testing",
    "file_name": "1010_missing_internet.json",
    "relative_file_path": "mud_files/manipulated_w_o_telling_names/1010_missing_internet.json",
    "device_under_test": "Edge Gateway",
    "scenario_title": "Missing Internet Connectivity (No System-Info Variant)",
    "manipulation_description": "Crucial rules enabling internet connectivity have been intentionally suppressed to simulate an under-permissive configuration, formatted as a 'no system info' (NTN) variant to support context-ablation analysis.",
    "expected_conformance_level": "Gold",
```

```
    "expected_functional_security_level": "Insecure",
    "expected_accuracy_class": "under_permissive",
    "notes": "No-system-info variant of S10.",
    "aliases": []
  },
  {
    "scenario_id": "S12_NTN",
    "variant_group_id": "S12",
    "dataset_kind": "manipulated_w_o_telling_names",
    "stage": "accuracy_testing",
    "file_name": "1012_missing_dns.json",
    "relative_file_path": "mud_files/manipulated_w_o_telling_names/1012_missing_dns.json",
    "device_under_test": "Edge Gateway",
    "scenario_title": "Missing DNS Connectivity (No System-Info Variant)",
    "manipulation_description": "Domain Name System (DNS) query rules have been systematically deleted to test connectivity dependency detection, provided herein as a 'no system info' (NTN) variant without explicit metadata.",
    "expected_conformance_level": "Gold",
    "expected_functional_security_level": "Secure",
    "expected_accuracy_class": "under_permissive",
    "notes": "No-system-info variant of S12.",
    "aliases": []
  },
  {
    "scenario_id": "S11_NTN",
    "variant_group_id": "S11",
    "dataset_kind": "manipulated_w_o_telling_names",
    "stage": "accuracy_testing",
    "file_name": "1011_half_a_standard.json",
    "relative_file_path": "mud_files/manipulated_w_o_telling_names/1011_half_a_standard.json",
    "device_under_test": "Vibration Sensor",
    "scenario_title": "Missing Half of a Standard (No System-Info Variant)",
    "manipulation_description": "A critical segment of the PTP standard rule set has been removed to assess the identification of incomplete configurations, structured as a 'no system info' (NTN) variant without descriptive fields.",
```

```
    "expected_conformance_level": "Gold",
    "expected_functional_security_level": "Insecure",
    "expected_accuracy_class": "under_permissive",
    "notes": "No-system-info variant of S11.",
    "aliases": []
  },
  {
    "scenario_id": "S13_NTN",
    "variant_group_id": "S13",
    "dataset_kind": "manipulated_w_o_telling_names",
    "stage": "accuracy_testing",
    "file_name": "1013_open_door.json",
    "relative_file_path": "mud_files/manipulated_w_o_telling_names/1013
_open_door.json",
    "device_under_test": "Edge Gateway",
    "scenario_title": "Open Door (No System-Info Variant)",
    "manipulation_description": "A broadly permissive inbound TCP port
443 rule has been integrated without IP restrictions, distributed as a
 'no system info' (NTN) variant to evaluate context-free over-
permissive detections.",
    "expected_conformance_level": "Gold",
    "expected_functional_security_level": "Insecure",
    "expected_accuracy_class": "over_permissive",
    "notes": "No-system-info variant of S13.",
    "aliases": []
  },
  {
    "scenario_id": "S14_NTN",
    "variant_group_id": "S14",
    "dataset_kind": "manipulated_w_o_telling_names",
    "stage": "accuracy_testing",
    "file_name": "1014_silent_data_exfil.json",
    "relative_file_path": "mud_files/manipulated_w_o_telling_names/1014
_silent_data_exfil.json",
    "device_under_test": "mqtt-bridge",
    "scenario_title": "Silent Data Exfiltration (No System-Info Variant)
",
    "manipulation_description": "An outbound data exfiltration channel
has been intentionally designed into the configuration, serving as a '
no system info' (NTN) variant to assess silent leakage detection
without aiding device metadata.",
```

```
    "expected_conformance_level": "Gold",
    "expected_functional_security_level": "Insecure",
    "expected_accuracy_class": "over_permissive",
    "notes": "No-system-info variant of S14.",
    "aliases": [
      "silent_data_exfiltration"
    ]
  },
  {
    "scenario_id": "S15_NTN",
    "variant_group_id": "S15",
    "dataset_kind": "manipulated_w_o_telling_names",
    "stage": "accuracy_testing",
    "file_name": "1015_implausible_connection.json",
    "relative_file_path": "mud_files/manipulated_w_o_telling_names/1015_implausible_connection.json",
    "device_under_test": "Safety Block",
    "scenario_title": "Implausible Internet Connection (No System-Info Variant)",
    "manipulation_description": "An arbitrary, high-risk public internet inbound access rule has been added, formulated as a 'no system info' (NTN) variant to evaluate implausible connection detection amidst a context-ablated dataset.",
    "expected_conformance_level": "Gold",
    "expected_functional_security_level": "Insecure",
    "expected_accuracy_class": "over_permissive",
    "notes": "No-system-info variant of S15.",
    "aliases": []
  },
  {
    "scenario_id": "S16_NTN",
    "variant_group_id": "S16",
    "dataset_kind": "manipulated_w_o_telling_names",
    "stage": "accuracy_testing",
    "file_name": "1016_lateral_movement.json",
    "relative_file_path": "mud_files/manipulated_w_o_telling_names/1016_lateral_movement.json",
    "device_under_test": "Edge Gateway",
    "scenario_title": "Lateral Movement Pivot (No System-Info Variant)",
    "manipulation_description": "A specialized SMB lateral movement rule has been introduced, intentionally stripping away the `system-info`
```

```
        field to act as a 'no system info' (NTN) variant for advanced network
        pivot simulations.",
3.0.579   "expected_conformance_level": "Gold",
3.0.580   "expected_functional_security_level": "Insecure",
3.0.581   "expected_accuracy_class": "over_permissive",
3.0.582   "notes": "No-system-info variant of S16.",
3.0.583   "aliases": []
3.0.584  },
3.0.585  {
3.0.586   "scenario_id": "S19",
3.0.587   "variant_group_id": "S19",
3.0.588   "dataset_kind": "manipulated",
3.0.589   "stage": "accuracy_testing",
3.0.590   "file_name": "119_local_http_usage.json",
3.0.591   "relative_file_path": "mud_files/manipulated/119_local_http_usage.
        json",
3.0.592   "device_under_test": "Edge Gateway",
3.0.593   "scenario_title": "Local http Usage, bound to local-networks",
3.0.594   "manipulation_description": "An overly permissive rule authorizing
        HTTP traffic on port 80 has been confined to local networks within the
         edge gateway configuration to assess baseline anomaly detection
        without explicit vulnerability libraries.",
3.0.595   "expected_conformance_level": "Platinum",
3.0.596   "expected_functional_security_level": "Insecure",
3.0.597   "expected_accuracy_class": "over_permissive",
3.0.598   "notes": null,
3.0.599   "aliases": []
3.0.600  },
3.0.601  {
3.0.602   "scenario_id": "S19:_NTN",
3.0.603   "variant_group_id": "S19_NTN",
3.0.604   "dataset_kind": "manipulated_w_o_telling_names",
3.0.605   "stage": "accuracy_testing",
3.0.606   "file_name": "1019_local_http_usage.json",
3.0.607   "relative_file_path": "mud_files/manipulated_w_o_telling_names/1019
        _local_http_usage.json",
3.0.608   "device_under_test": "Edge Gateway",
3.0.609   "scenario_title": "Local http Usage, bound to local-networks",
3.0.610   "manipulation_description": "An overly permissive rule authorizing
        HTTP traffic on port 80 has been confined to local networks, formatted
```

```
        specifically as a 'no system info' (NTN) variant for context-ablation
        evaluations.",
3.0.611   "expected_conformance_level": "Gold",
3.0.612   "expected_functional_security_level": "Insecure",
3.0.613   "expected_accuracy_class": "over_permissive",
3.0.614   "notes": "No telling names variant of S19.",
3.0.615   "aliases": []
3.0.616 },
3.0.617 {
3.0.618   "scenario_id": "S20",
3.0.619   "variant_group_id": "S20",
3.0.620   "dataset_kind": "manipulated",
3.0.621   "stage": "accuracy_testing",
3.0.622   "file_name": "120_covered_http_usage.json",
3.0.623   "relative_file_path": "mud_files/manipulated/120_covered_http_usage.
        json",
3.0.624   "device_under_test": "Edge Gateway",
3.0.625   "scenario_title": "HTTP Usage, open to any address",
3.0.626   "manipulation_description": "A widely expansive rule facilitating
        HTTP traffic on port 80 from completely unrestricted global addresses
        has been introduced to the edge gateway to probe the detection rate of
        insecure, broad-spectrum exposures.",
3.0.627   "expected_conformance_level": "Platinum",
3.0.628   "expected_functional_security_level": "Insecure",
3.0.629   "expected_accuracy_class": "over_permissive",
3.0.630   "notes": null,
3.0.631   "aliases": []
3.0.632 },
3.0.633 {
3.0.634   "scenario_id": "S20_NTN",
3.0.635   "variant_group_id": "S20_NTN",
3.0.636   "dataset_kind": "manipulated_w_o_telling_names",
3.0.637   "stage": "accuracy_testing",
3.0.638   "file_name": "1020_covered_http_usage.json",
3.0.639   "relative_file_path": "mud_files/manipulated_w_o_telling_names/1020
        _covered_http_usage.json",
3.0.640   "device_under_test": "Edge Gateway",
3.0.641   "scenario_title": "HTTP Usage, open to any address",
3.0.642   "manipulation_description": "A widely expansive rule facilitating
        HTTP traffic on port 80 from unrestricted global addresses has been
```

```
          included as a 'no system info' (NTN) variant to evaluate the detection
           of excessive permissions.",
3.0.643       "expected_conformance_level": "Gold",
3.0.644       "expected_functional_security_level": "Insecure",
3.0.645       "expected_accuracy_class": "over_permissive",
3.0.646       "notes": "No telling names variant of S20.",
3.0.647       "aliases": []
3.0.648     }
3.0.649   ]
3.0.650 }
3.0.651 }
```

Declaration of Authorship

‘I hereby declare,

- that I have written this thesis independently;
- that I have written the thesis using only the aids listed in the index;
- that all parts of the thesis produced with the help of aids (incl. AI-Tools) have been precisely declared;
- that I have mentioned all sources used and cited them correctly according to established academic citation rules; this also includes the comprehensible disclosure of all personal publications;
- that I have acquired all immaterial rights to any materials I may have used, such as images or graphics, or that these materials were originally created by myself;
- that I am aware of the legal provisions regarding the publication and dissemination of parts or the entire thesis and that I comply with them accordingly;
- that I am aware that the University will prosecute a violation of this Declaration of Authorship and that disciplinary as well as criminal consequences may result, which may lead to expulsion from the University or to the withdrawal of my title.’

By submitting this thesis, I confirm through my conclusive action that I am submitting the statutory declaration, that I have read and understood it, and that it is true.

St. Gallen, Mai 19, 2026 Signature:

Declaration of Auxiliary Aids

The following auxiliary tools were used in the completion of this thesis:

Aid	**Usage**	**Affected Parts**
Google Gemini 2.5 (Deep Research)	Finding papers	Literature review
Google NotebookLM	Literature review and structuring the presentation of the literature review	Literature review
Perplexity AI	Finding specific sources	Complete paper
Google Gemini 2.5 (Deep Thinking)	Structuring the thesis	Complete paper
GitHub Copilot in VS Code (AUTO)	Coding and refactoring code	Code
GitHub Copilot in VS Code (Gemini 3.1 Pro)	Turning bullet points and text into academic prose	Complete paper
Cursor AI (Gemini 3.1 Pro)	Locating narrative gaps in the thesis and turning bullet points and text into academic prose	Complete paper