TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Adeyimika Agboola Adetunji 245205IVSM

Extending Sigma Detection Rules for Cyber-Physical Systems

Master's Thesis

Supervisor: Muaan Ur Rehman
MSc

Tallinn 2026

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Adeyimika Agboola Adetunji 245205IVSM

Sigma Tuvastusreeglite Laiendamine Küberfüüsikalistele Süsteemidele

Magistritöö

Juhendaja: Muaan Ur Rehman
MSc

Tallinn 2026

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis and that this thesis has not been presented for examination or submitted for defense anywhere else. All used materials, references to the literature, and work of others have been cited.

Author: Adeyimika Agboola Adetunji

21.04.2026

Abstract

This thesis investigates how Sigma style portable detection engineering can be extended to cyber physical system telemetry. The work focuses on the problem that detection logic for industrial protocols is often implemented in tool specific scripts, parser specific workflows, or backend native rules, which makes such detections difficult to reuse, compare, and translate across monitoring environments. To address this problem, the thesis proposes a Sigma compatible extension for Modbus telemetry, referred to as Sigma CPS Modbus, and evaluates its feasibility through a working prototype.

The study follows a design science research approach. First, the requirements for representing Modbus specific detection logic in a readable and structured form are identified. Based on these requirements, a normalized Modbus event model and a Sigma compatible rule schema are designed. The schema preserves the general Sigma rule structure while extending the field vocabulary to support Modbus aware concepts such as function codes, operation type, address ranges, response status, and simple repeated event conditions. A prototype is then implemented in Python to load, validate, translate, and execute Sigma CPS Modbus rules over normalized telemetry.

The evaluation is performed on controlled normalized Modbus event sets representing four detection scenarios: unauthorized write requests from non approved sources, writes to protected register ranges, write requests resulting in exception responses, and repeated write activity from a single source within a defined time window. The prototype successfully processes all implemented rule types and correctly identifies the intended events in each scenario. The results show that Modbus detection logic can be authored in a Sigma compatible format and translated into executable logic in a systematic and readable way.

The main contribution of the thesis is a proof of concept demonstrating that a portable rule layer can be introduced between normalized Modbus telemetry and backend specific execution logic. The work does not aim to replace packet parsers or provide a full production

ready SIEM integration. Instead, it shows that backend neutral authoring of Modbus detections is feasible and that the remaining step toward operational deployment is mainly target specific mapping. The findings support the conclusion that Sigma compatible rule design can be extended beyond traditional IT logs and into selected cyber physical system telemetry contexts.

The thesis is in English and contains 62 pages of text, 8 chapters, 11 figures, 11 tables.

Annotatsioon

Sigma Tuvastusreeglite Laiendamine Küberfüüsikalistele Süsteemidele

Käesolev lõputöö uurib, kuidas Sigma laadset portatiivset tuvastusreeglite koostamise lähenemist saab laiendada küberfüüsikaliste süsteemide telemeetriale. Töö keskendub probleemile, et tööstusprotokollide tuvastusloogika realiseeritakse sageli tööriistaspetsiifiliste skriptide, parserispetsiifiliste töövoogude või platvormipõhiste reeglitena, mistõttu on selliseid tuvastusi keeruline taaskasutada, võrrelda ja erinevate seirekeskkondade vahel tõlkida. Selle probleemi lahendamiseks pakub töö välja Modbusi telemeetriale mõeldud Sigma ühilduva laienduse, mida nimetatakse Sigma CPS Modbus lähenemiseks, ning hindab selle teostatavust töötava prototüübi abil.

Uurimus järgib disainiteaduslikku metoodikat. Esmalt määratletakse nõuded, kuidas esitada Modbus spetsiifilist tuvastusloogikat loetaval ja struktureeritud kujul. Nende nõuete põhjal kavandatakse normaliseeritud Modbusi sündmusmudel ning Sigma ühilduv reegliskeem. Skeem säilitab Sigma üldise reeglistruktuuri, kuid laiendab väljade sõnavara, et toetada Modbusi põhiseid mõisteid, nagu funktsioonikoodid, operatsioonitüüp, aadressivahemikud, vastuse staatus ja lihtsad korduvsündmuste tingimused. Seejärel implementeeritakse Pythonis prototüüp, mis laadib, valideerib, tõlgib ja käivitab Sigma CPS Modbus reegleid normaliseeritud telemeetria peal.

Hindamine viiakse läbi kontrollitud normaliseeritud Modbusi sündmushulkadel, mis kirjeldavad nelja tuvastusstsenaariumi: volitamata kirjutuspäringud mittetunnustatud allikatest, kirjutused kaitstud registrivahemikesse, erindvastusega lõppevad kirjutuspäringud ning korduv kirjutusaktiivsus ühelt allikalt etteantud ajavahemiku jooksul. Prototüüp töötleb edukalt kõiki implementeeritud reeglitüüpe ja tuvastab igas stsenaariumis õiged sündmused. Tulemused näitavad, et Modbusi tuvastusloogikat on võimalik kirjeldada Sigma ühilduvas formaadis ja tõlkida see süstemaatiliselt ning loetavalt käivitatavaks loogikaks.

Töö peamine panus on kontseptsiooni tõestus, mis näitab, et normaliseeritud Modbusi telemeetria ja taustsüsteemispetsiifilise käivitusloogika vahele on võimalik lisada portatiivne reeglikihistus. Töö eesmärk ei ole asendada paketiparsereid ega pakkuda täielikult tootmiskõlblikku SIEM integratsiooni. Selle asemel näitab töö, et Modbusi tuvastuste taustsüsteemist sõltumatu kirjeldamine on teostatav ning järgmine samm praktilise kasutuse suunas on peamiselt sihtplatvormipõhine vastendamine. Tulemused toetavad järeldust, et Sigma ühilduvat reeglidisaini on võimalik laiendada traditsioonilistest IT logidest valitud küberfüüsikaliste süsteemide telemeetria konteksti.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 62 leheküljel, 8 peatükki, 11 joonist, 11 tabelit.

List of abbreviations and terms

CPS	Küberfüüsikaline süsteem (*cyber physical System*)
ICS	Tööstuslik juhtimissüsteem (*Industrial Control System*)
OT	Operatsioonitehnoloogia (*Operational Technology*)
IT	Infotehnoloogia (*Information Technology*)
PLC	Programmeeritav loogikakontroller (*Programmable Logic Controller*)
HMI	Inimese-masina liides (*Human-Machine Interface*)
SIEM	Turbeinfo ja sündmuste haldus (*Security Information and Event Management*)
IDS	Sissetungituvastussüsteem (*Intrusion Detection System*)
YAML	YAML Ain't Markup Language
TCP	Edastuse juhtimisprotokoll (*Transmission Control Protocol*)
PDU	Protokolliandmeüksus (*Protocol Data Unit*)
PCAP	Pakettide jäädvustus (*Packet Capture*)
API	Rakendusliides (*Application Programming Interface*)
CISA	Küberturbe ja taristukaitse agentuur (*Cybersecurity and Infrastructure Security Agency*)
NIST	Riiklik standardite ja tehnoloogia instituut (*National Institute of Standards and Technology*)
DSRM	Disainiteaduslik uurimismetoodika (*Design Science Research Methodology*)

Table of contents

1 Introduction 13
1.1 Background and motivation 14
1.2 Research problem 14
1.3 Research questions 15
1.4 Research hypothesis 16
1.5 Scope and limitations 16
1.6 Thesis structure 17
2 Background and Literature Review 18
2.1 Portable detection engineering and Sigma 18
2.2 cyber physical systems and OT telemetry 19
2.3 Modbus as the selected protocol 20
2.4 Existing CPS/ICS detection approaches 21
2.5 Protocol parsing and telemetry normalization 23
2.6 Research gap and thesis positioning 24
3 Methodology 26
3.1 Research method 26
3.2 Research design 26
3.3 Materials, datasets, and tools 27
3.4 Design and implementation process 28
3.5 Evaluation criteria 29
4 Design of the Sigma CPS Modbus Extension 30
4.1 Design goals and requirements 30
4.2 Normalized Modbus event model 33
4.3 Extension of the Sigma schema for Modbus telemetry 36
4.4 Rule structure and field definitions 38
4.5 Example Sigma CPS Modbus rules 40
5 Prototype Implementation 44

5.1 System architecture ... 44
5.2 Input and output design ... 47
5.3 Rule parser and validator ... 48
5.4 Translation workflow ... 49
5.5 Execution over normalized Modbus telemetry ... 51
5.6 Implemented rules and observed prototype behaviour ... 55
5.7 Prototype limitations ... 57
6 Evaluation and Results ... 58
6.1 Evaluation setup ... 58
6.2 Detection scenarios ... 59
6.3 Results ... 60
6.4 Analysis of findings ... 65
7 Discussion ... 67
7.1 Interpretation of results ... 67
7.2 Readability and logical portability of the approach ... 68
7.3 Practical usefulness ... 68
7.4 Limitations and threats to validity ... 69
7.5 Future directions ... 70
8 Summary ... 72
References ... 75
Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis ... 78
Appendix 2 – Function-code mapping and normalization details ... 79
Appendix 3 – Prototype artifacts and selected outputs ... 81

List of figures

Figure 1. Normalized Modbus event model grouped by field category. 34

Figure 2. Overall Sigma CPS Modbus prototype architecture. 46

Figure 3. Directory structure of the Sigma CPS Modbus prototype. 47

Figure 4. Rule translation workflow from YAML input to executable predicate........ 50

Figure 5. Aggregation workflow for repeated event detection................................ 52

Figure 6. Example of Modbus/TCP traffic observed in Wireshark. Source: author's screenshot based on ICS_PCAPS MODBUS/TCP#1 packet captures [26]. 54

Figure 7. Detailed Modbus packet decode in Wireshark showing transaction level and function level fields. Source: author's screenshot based on ICS_-PCAPS MODBUS/TCP#1 packet captures [26]. 55

Figure 8. Prototype execution of the unauthorized Modbus write rule, showing rule definition, normalized input events, and resulting matches...................... 61

Figure 9. Prototype execution of the sensitive register range rule, showing rule definition, normalized event data, and resulting match. 62

Figure 10. Prototype execution of the write exception rule, showing rule definition, normalized event set, resulting match, and automated test result.............. 63

Figure 11. Prototype execution of the aggregation based excessive write rule, demonstrating threshold based repeated event detection within a one minute time window. 64

List of tables

Table 1. Comparison of existing CPS/ICS detection approaches and Sigma CPS Modbus positioning. 23

Table 2. Design goals of the Sigma CPS Modbus extension.......... 33

Table 3. Core normalized Modbus event fields used by the prototype. 35

Table 4. Supported field operators in the prototype.......... 39

Table 5. Core function code mapping used in the prototype implementation.......... 54

Table 6. Prototype modules and responsibilities.......... 56

Table 7. Evaluation scenarios and expected outcomes.......... 60

Table 8. Summary of prototype rule execution results.......... 60

Table 9. Automated test summary.......... 65

Table 10. Function-code mapping used by the Sigma-CPS Modbus normalizer. 79

Table 11. Common Modbus exception codes relevant to the prototype. 80

1 Introduction

cyber physical systems (CPS) combine computational components, communication networks, and physical processes. In practice, this broad category includes operational technology (OT) environments such as industrial control systems, programmable logic controllers, supervisory systems, and field devices that directly monitor or influence physical processes. Because these systems interact with the physical world, their security requirements differ from those of conventional enterprise information technology (IT) systems. In OT environments, availability, timing, reliability, and safety are often as important as confidentiality and integrity. NIST emphasizes that OT environments operate under distinct performance, reliability, and safety constraints, which means that security practices from enterprise IT cannot be transferred directly without adaptation [1].

In enterprise detection engineering, Sigma has become a widely used open format for expressing detection logic in a readable YAML structure that can later be converted into backend specific queries and detections. The Sigma ecosystem is supported by an official rule specification and by pySigma, a Python library for parsing, transforming, and converting Sigma rules [2, 3]. This separation between rule logic and implementation backend has contributed to Sigma's practical value in log centric enterprise monitoring. However, CPS environments often expose security relevant behaviour through protocol semantics, message structure, and communication patterns rather than only through standardized host logs [4, 5]. Extending Sigma toward CPS therefore requires adapting a log oriented abstraction model to protocol centric telemetry.

This thesis addresses that mismatch by investigating how Sigma style portable detection engineering can be extended toward CPS telemetry. The overall topic remains the extension of Sigma detection rules to cyber physical systems, but the practical realization of the work is narrowed to Modbus based telemetry. This narrowing is deliberate and methodological. It allows the thesis to remain focused on the main research contribution while still evaluating the idea in one representative protocol domain.

1.1 Background and motivation

Portable detection content is valuable because it allows detection logic to be reviewed, shared, adapted, and translated across different security tools. Sigma was created precisely for that purpose in enterprise environments, where detections are commonly expressed against relatively stable event fields and then converted into platform specific implementations [2]. pySigma further operationalizes this model by providing parsing, transformation, validation, and conversion workflows for Sigma rules [3, 6, 7].

In CPS and OT environments, however, security monitoring frequently depends on protocol aware telemetry. For Modbus in particular, security relevant meaning is encoded in application layer function codes, register addressing, read and write operations, and communication patterns defined by the protocol specification [8]. The Modbus standard explicitly documents these function level semantics, which makes the protocol suitable for structured rule based detections such as identifying unauthorized write operations or suspicious access to sensitive address ranges. At the same time, prior research has shown that Modbus traffic often exhibits highly structured and repetitive behaviour, which means that protocol aware detection can benefit from more expressive representations than generic log matching alone [9, 10].

The motivation for this thesis is therefore both practical and research driven. Practically, CPS detections are often implemented as tool specific scripts, parser specific workflows, or engine specific signatures, which makes them harder to reuse across heterogeneous monitoring stacks [11, 12, 13, 14]. Research wise, there is a clear opportunity to test whether the portability principles behind Sigma can be extended into protocol centric monitoring without losing clarity and readability. Existing parser ecosystems such as CISA's ICSNPP illustrate a parser first approach for industrial protocols, and related Modbus focused tooling suggests how parsed telemetry can be made available in more structured form for downstream analysis [15, 16, 14].

1.2 Research problem

The research problem addressed in this thesis is the absence of a standardized, portable, and human readable rule format for expressing detection logic over CPS protocol telemetry.

Although Sigma provides a mature and widely recognized abstraction for enterprise detection engineering, its original use cases are centered on traditional IT data sources rather than industrial protocol semantics [2]. As a result, CPS detections are still commonly implemented through backend specific rules, custom scripts, or tightly coupled parser logic instead of reusable rule abstractions [4, 5].

This limitation creates several practical consequences. First, detection knowledge becomes difficult to reuse across monitoring tools and organizations. Second, translation and automation are constrained because there is no common schema for protocol aware rule content. Third, detection engineering in IT and OT remains more fragmented than necessary, even when similar operational goals exist. In the case of Modbus, these challenges are especially visible because the protocol exposes clearly defined application semantics that are meaningful for detection, but those semantics are typically handled at the parser or script level rather than in a portable rule layer [8, 9, 10].

Accordingly, this thesis investigates whether a Sigma compatible extension can be designed for CPS telemetry and demonstrated in a focused Modbus implementation. The problem is not whether Modbus attacks can be detected at all, since prior research and existing tooling show that they can [9, 10, 17]. The actual problem is whether such detection logic can be represented in a structured, portable, and implementation independent way that remains readable to analysts while still being translatable into practical detections.

1.3 Research questions

To address the research problem, this thesis is guided by the following research questions:

1. How can the Sigma rule schema be extended to represent Modbus telemetry while preserving the readability and simplicity that characterize Sigma?
2. How can Sigma CPS Modbus rules be translated into executable detection logic through a prototype translation workflow?
3. How effective and practically useful is the proposed approach for representing and detecting representative Modbus misuse scenarios?

These questions refine the broader CPS detection problem into a scope that is feasible for this thesis while remaining aligned with the original research direction of extending Sigma

detection rules toward cyber physical system telemetry.

1.4 Research hypothesis

This thesis is based on the hypothesis that extending Sigma with Modbus aware telemetry abstractions and protocol specific detection primitives can improve the standardization, portability, and practical usability of CPS detection rules without sacrificing the readability that makes Sigma valuable in enterprise detection engineering. The hypothesis assumes that Modbus telemetry can be normalized into sufficiently stable rule fields and that those fields can support translation into executable detections after protocol parsing has already taken place. This assumption is supported by parser first industrial monitoring approaches and by prior Modbus focused intrusion detection research showing that protocol level features are meaningful for security monitoring [15, 16, 9, 10].

1.5 Scope and limitations

The topic of this thesis remains the extension of Sigma detection rules to cyber physical systems. However, the design, implementation, and evaluation are deliberately narrowed to Modbus as the representative protocol for the practical part of the study. This refinement is methodological rather than conceptual: CPS environments include many protocols and communication models, but implementing and validating a multi protocol framework within the limits of a single thesis would introduce substantial complexity without proportionate benefit to the core research objective. Modbus was selected because it is a widely used industrial protocol, its application semantics are formally documented, and it supports clearly identifiable security relevant operations such as read and write function codes [8, 18].

The thesis therefore focuses on the rule representation layer and a prototype translator for normalized Modbus telemetry. It does not aim to replace protocol parsers, build a full production ready detection platform, or solve the broader problem of process aware anomaly detection in OT environments. Instead, the study assumes that protocol aware parsing has already taken place and that the resulting structured telemetry can be consumed by a Sigma compatible rule layer. This architectural assumption is consistent with parser first industrial monitoring approaches reflected in the ICSNPP ecosystem and related

Modbus tooling [15, 16, 14].

The empirical evaluation is also limited. Rather than claiming comprehensive coverage of all CPS attack classes, the thesis concentrates on representative Modbus misuse scenarios such as unauthorized write operations, suspicious register access, exception generating writes, and repeated write behaviour. The implemented prototype is evaluated primarily on controlled normalized event sets and automated tests designed to verify correctness of the rule processing pipeline. Public Modbus datasets such as CIC Modbus 2023 are relevant for future external validation, but large scale parser to dataset integration is outside the present implementation scope [19].

1.6 Thesis structure

The remainder of this thesis is organized as follows. Chapter 2 reviews the literature on portable detection engineering, CPS and OT telemetry, Modbus related intrusion detection, protocol parsing, and related detection tooling. Chapter 3 presents the research methodology and explains the design science approach used in the study. Chapter 4 introduces the proposed Sigma CPS extension for Modbus telemetry, including its design goals, schema elements, and example rules. Chapter 5 describes the prototype implementation and translation workflow. Chapter 6 presents the evaluation setup, scenarios, and results. Chapter 7 discusses the findings, practical usefulness, limitations, and future directions of the work. Finally, Chapter 8 concludes the thesis by summarizing the contribution and answering the research questions.

2 Background and Literature Review

This chapter reviews the work most relevant to the thesis and establishes the theoretical and technical background for the proposed Sigma CPS Modbus approach. The discussion begins with portable detection engineering and Sigma, then examines CPS and OT telemetry, Modbus as the selected protocol, existing industrial detection approaches, protocol parsing and normalization, and finally the research gap addressed by this thesis.

2.1 Portable detection engineering and Sigma

Portable detection engineering aims to separate detection logic from the specific query languages, schemas, and operational constraints of individual security platforms. In conventional enterprise security monitoring, this separation is valuable because it improves reuse, collaboration, and maintainability: a detection can be defined once at an abstract level and later converted into platform specific implementations. Sigma is one of the most established open standards pursuing this goal. The official Sigma rules specification defines a structured YAML rule format, while the broader Sigma ecosystem provides conventions for rule metadata, detection fields, and conditions. In practice, this makes Sigma a backend neutral way to represent detection logic rather than a detection engine by itself [2, 20].

The practical usefulness of Sigma is reinforced by pySigma, which provides a programmatic framework for parsing, transforming, validating, and converting Sigma rules. In the pySigma architecture, backends are responsible for producing target queries, while processing pipelines handle differences between Sigma rules and the target data model, such as field renaming and value normalization [3, 6, 7]. This distinction is especially relevant for the present thesis because CPS telemetry often differs substantially from standard enterprise log schemas. In other words, the Sigma ecosystem already contains the architectural idea needed for this research: portable rule logic can remain stable while transformations adapt the rule to a specific telemetry model.

Sigma also includes a correlation oriented specification for multi event conditions beyond simple single event matching [21]. While the current thesis does not implement the full breadth of Sigma correlation functionality, this is still important background because CPS detections often depend on repeated actions, rates, or short sequences of behaviour rather than on isolated events. Sigma is therefore a suitable conceptual starting point even if protocol specific extensions are still required.

At the same time, Sigma's original center of gravity remains enterprise IT. Its best supported use cases are still log centric sources such as Windows event logs, Sysmon, cloud telemetry, and endpoint detections. That does not make Sigma unsuitable for CPS, but it does mean that its default assumptions were not designed around industrial protocols, device roles, or control oriented communication semantics. This is the key tension that motivates the thesis: Sigma provides a compelling abstraction model, yet CPS telemetry presents data characteristics that are not naturally first class citizens in that model.

2.2 cyber physical systems and OT telemetry

cyber physical systems combine computational control, networked communication, and physical processes. In operational technology environments, this includes systems that monitor or directly influence devices, industrial processes, and control events. NIST emphasizes that OT differs from traditional IT because of its distinctive performance, reliability, and safety requirements. OT systems and devices directly monitor or control devices, processes, and events in the physical environment, which means that monitoring and detection in OT cannot simply reuse enterprise assumptions without adaptation [1].

This difference is reflected in the nature of telemetry itself. In enterprise environments, analysts often work with host generated events that already arrive in relatively structured formats. In CPS and OT environments, important security evidence is frequently embedded in network protocol traffic, controller communications, command semantics, timing regularity, and communication roles between devices such as HMIs, PLCs, and engineering stations. Mitchell and Chen's survey of intrusion detection for cyber physical systems highlights that CPS detection techniques must often combine cyber observations with behavioural or process aware signals rather than relying only on traditional host log abstractions [4]. Similarly, Hu et al. classify ICS IDS approaches into protocol analysis

based, traffic mining based, and control process analysis based families, which illustrates how industrial detection frequently depends on semantics that are not native to conventional SIEM event models [5].

This has two implications for the thesis. First, CPS detection engineering often requires more protocol awareness than ordinary log matching. Second, a portable rule format for CPS cannot rely exclusively on assumptions inherited from host or SIEM telemetry. Instead, it must be able to represent fields derived from protocol parsing and, where necessary, preserve the semantics of device interactions and operation types. These characteristics make CPS telemetry a useful but demanding test case for extending Sigma style abstraction.

2.3 Modbus as the selected protocol

Although the overall thesis addresses Sigma extensions for CPS, the practical realization is narrowed to Modbus. This is a deliberate methodological refinement rather than a change in topic. CPS covers many protocols and communication settings, but implementing and evaluating several protocol families within one thesis would substantially increase complexity at the schema, parser, and validation levels. The narrowed Modbus focus therefore preserves the core research objective while making the artifact design and evaluation feasible.

Modbus is a suitable representative protocol for several reasons. First, it remains one of the most widely recognized industrial communication protocols. Second, its application semantics are formally documented by the Modbus Application Protocol Specification, which defines the protocol data unit, addressing model, and function codes. These function codes expose behaviour that is directly useful for rule based detection, including distinctions between read operations, write operations, diagnostic functions, and other command types [8, 18]. For example, detection logic can be tied to write capable functions, unexpected register access, or unusual combinations of request origin and operation type.

A further reason for selecting Modbus is that the research literature already demonstrates its relevance to intrusion detection. Goldenberg and Wool showed that Modbus/TCP traffic is often highly periodic and structured, which makes it suitable for protocol aware and behaviour aware monitoring [9]. Morris et al. further demonstrated that deterministic

intrusion detection rules can be derived directly from Modbus protocol semantics and vulnerability knowledge [10]. Therefore, Modbus offers a strong bridge between two worlds that are both relevant to this thesis: it supports explicit field level rule conditions, but it also illustrates why richer CPS detections may involve structure and timing rather than only isolated field matches.

2.4 Existing CPS/ICS detection approaches

Existing CPS and ICS detection approaches are diverse, but they can be broadly grouped into signature based, rule based, model based, and anomaly based families. Signature based systems remain widely used in operational settings because they are interpretable and easier to validate operationally. In the industrial domain, this often appears as engine specific signatures or handcrafted rules aimed at specific protocol abuses. Morris et al., for example, proposed deterministic intrusion detection rules for Modbus/TCP and Modbus serial environments, deriving signatures from protocol vulnerabilities [10]. This line of work is important because it demonstrates that useful detection knowledge can indeed be expressed for Modbus, even if the resulting rule content is not portable across tooling by default.

Model based and anomaly based approaches have also been prominent in ICS research. Goldenberg and Wool modeled Modbus/TCP traffic around the highly periodic communication patterns between HMIs and PLCs [9], while more recent work has explored open source anomaly detection for Modbus and classification based analysis over control system traffic [17, 22]. These approaches are often effective because industrial traffic is repetitive and role constrained, but they may also depend on site specific training, finely tuned baselines, or custom parsing logic. As a result, they do not by themselves solve the problem of expressing detection knowledge in a portable rule abstraction.

Operational CPS monitoring also commonly uses tool specific detection forms. Zeek exposes parsed Modbus telemetry through its scripting and logging interface, making scripted detection over Modbus records possible [14]. Zeek's generic Notice framework can then be used to generate alerts from such analysis logic [23]. Suricata and Snort provide rule based inspection, but rule logic remains tightly tied to the capabilities and syntax of the selected engine [11]. Digital Bond's Quickdraw repositories illustrate this

clearly: Quickdraw Snort is explicitly presented as a set of ICS IDS/IPS rules for Snort, and Quickdraw Suricata as a set of ICS IDS rules for Suricata [12, 13]. These approaches show that industrial detections can be operationalized successfully, but they also reveal fragmentation: detection content remains difficult to share and reuse across heterogeneous CPS monitoring stacks without a common abstraction layer.

This fragmentation is not merely a matter of implementation style. In engine specific approaches such as Suricata, Snort, and Quickdraw, the detection logic is tightly coupled to the syntax, parsing model, and feature set of the chosen engine. In Zeek, the analyst gains rich access to parsed telemetry, but the resulting logic still remains tied to Zeek's scripting and event model rather than existing as reusable backend neutral rule content. In anomaly based and model based approaches, the logic may depend on local baselines, custom feature engineering, or training assumptions, which limits reuse across environments even when the underlying detection idea is sound. Consequently, these approaches demonstrate that industrial detections can be built, but they do not provide a portable rule representation that can be shared and translated systematically across tools.

Table 1 summarizes the main differences between these approaches from the perspective of this thesis.

Table 1. Comparison of existing CPS/ICS detection approaches and Sigma CPS Modbus positioning.

Approach	Main strength	Main limitation for this thesis
Engine specific signatures (Snort/Suricata/Quickdraw)	Operationally deployable and interpretable	Tied to engine syntax and parser capabilities; weak portability across tools
Zeek scripting and Notice based logic	Rich access to parsed protocol events and logs	Detection logic remains scripting specific rather than portable rule content
Protocol specific anomaly or model based IDS	Can capture structured or repetitive industrial behaviour	Often depends on site specific baselines, custom features, or narrow implementations
Sigma style abstraction	Readable, structured, backend neutral rule representation	Needs protocol aware normalization and backend mapping for CPS use cases

From the perspective of this thesis, the key observation is that existing CPS detection research already contains valuable detection logic, but that logic is fragmented across papers, datasets, IDS engines, scripts, and parser specific workflows. In other words, the field does not lack detection ideas; it lacks a standard, readable, and implementation independent representation layer that could make such ideas easier to compare, share, and translate.

2.5 Protocol parsing and telemetry normalization

Protocol parsing is a prerequisite for portable detection in CPS because protocol semantics must first be converted into stable, structured fields. In the enterprise world, this role is often played by the logging source itself. In CPS, however, many security relevant features are only available after decoding the protocol message and extracting its specific semantics. The ICSNPP ecosystem developed by CISA illustrates a parser first approach in which industrial protocol traffic is decoded into structured telemetry before higher level detection logic is applied [15].

This pattern is also relevant in the Modbus case. Related Modbus focused tooling in the ICSNPP ecosystem suggests how default telemetry can be enriched for more detailed monitoring workflows [16]. Zeek's own Modbus scripting interface documents the events and records that expose parsed Modbus data to the logging framework, showing that parsed Modbus information is meant to become structured log content rather than remain only packet level data [14]. Rather than replacing the base parser, such approaches enrich the telemetry representation so that finer grained detection becomes possible. This is highly relevant to the thesis architecture. If Modbus traffic can be parsed into structured logs containing consistent fields, then a Sigma compatible rule layer can operate over those normalized fields instead of raw packets. In that sense, protocol parsing and rule portability are complementary rather than competing concerns.

The same logic is reflected in pySigma's processing pipeline model. There, transformations adapt rules to the target data model while leaving the high level detection logic intact [6]. Applied to CPS, a similar architecture would mean that protocol parsers produce normalized Modbus telemetry, a Sigma CPS rule describes the detection at an abstract level, and a translator or backend emits executable logic for the selected monitoring environment. This layered approach is one of the strongest conceptual foundations for the solution proposed in this thesis.

However, richer and more consistent telemetry alone does not solve the core problem addressed in this work. Even when protocol parsers produce structured Modbus logs, the detection logic built on top of those logs often remains tied to parser specific scripts, engine native rules, or local implementation choices. This leads directly to the research gap addressed in the next section: the absence of a portable rule layer above normalized CPS telemetry.

2.6 Research gap and thesis positioning

The reviewed literature reveals an important gap. On one side, Sigma and pySigma show that portable, readable, and transformable detection rules are feasible and operationally valuable in enterprise security monitoring [2, 3, 6]. On the other side, CPS and ICS research shows that industrial detections often depend on protocol specific semantics, parser derived fields, and communication patterns, particularly in protocols such as Modbus [4, 5,

9]. What is missing between these two bodies of work is a structured rule representation that brings Sigma style portability into CPS telemetry without discarding the protocol awareness required by OT monitoring.

Accordingly, this thesis is positioned neither as a new protocol parser nor as a comprehensive industrial anomaly detection system. Its contribution is narrower and more architectural: to design and evaluate a Sigma compatible extension for Modbus telemetry and to demonstrate, through a prototype translator, that Modbus detection logic can be represented in a readable, backend neutral, and executable way. This positioning aligns with the research problem defined in Chapter 1 and keeps the work focused on the rule representation layer rather than on full stack industrial security monitoring.

The value of this positioning is twofold. First, it creates a bridge between mature enterprise detection engineering practices and the protocol centric realities of OT monitoring. Second, it keeps the thesis contribution focused and feasible: instead of claiming to solve CPS detection in general, it tests the viability of a rule abstraction layer in one representative protocol domain. If successful, this provides evidence that Sigma style portability can be extended beyond traditional IT logs and into selected CPS telemetry contexts.

This chapter therefore establishes the basis for the next stages of the thesis. The reviewed sources support the need for protocol aware normalization, confirm the practical relevance of Modbus security monitoring, and show that the central unresolved issue is not detection capability alone but the lack of a portable and reusable rule representation above tool specific implementations.

3 Methodology

This chapter presents the methodology used in the thesis and explains how the proposed Sigma CPS Modbus approach was designed, implemented, and evaluated. Because the work aims to produce and assess a practical technical artifact, the methodological focus is not only analytical but also constructive.

3.1 Research method

This thesis follows the Design Science Research Methodology because its objective is to design, implement, and evaluate a technical artifact that addresses a concrete engineering problem. In this case, the artifact consists of two connected parts: a Sigma compatible extension for Modbus telemetry and a prototype translator that converts the extended rules into executable detection logic [24].

Design Science is appropriate because the thesis is not limited to identifying a gap in existing work. It proposes a solution, implements that solution in prototype form, and evaluates whether the result is useful within the defined scope of the study. The research therefore combines conceptual design, technical development, and empirical testing. This makes Design Science a suitable methodological fit, since the intended outcome is both a research contribution and a working proof of concept.

3.2 Research design

The research design consists of four main stages.

The first stage is requirements analysis. This stage identifies which parts of existing Sigma rules are sufficient and which parts are missing when the target data source is Modbus telemetry rather than conventional IT logs. The analysis focuses on protocol level semantics such as function codes, register addresses, source and destination information, operation types, and simple repeated communication behaviour. The purpose of this stage is to

identify the minimum set of concepts that a Sigma compatible Modbus rule format must support.

The second stage is schema design. Based on the identified requirements, the thesis defines a Sigma compatible extension for Modbus telemetry. The goal is to preserve the readability and structure of Sigma while adding the fields and detection concepts needed to express Modbus specific behaviour. The design does not attempt to replace protocol parsers or create a new detection engine. Its purpose is to define a portable rule representation layer above normalized Modbus data.

The third stage is prototype implementation. A proof of concept translator is developed to read Sigma CPS Modbus rules and convert them into executable detection logic. The implementation is intended to demonstrate feasibility rather than production readiness. It shows that the proposed rule abstraction can be parsed, interpreted, and applied to structured Modbus telemetry in a repeatable way.

The fourth stage is evaluation. The proposed approach is tested on representative Modbus detection scenarios in order to assess both technical usefulness and engineering usefulness. The evaluation examines whether the designed schema can express relevant detections, whether the translator produces usable outputs, and whether the resulting detections work on selected Modbus data.

3.3 Materials, datasets, and tools

The thesis uses normalized Modbus telemetry as the main data source for rule design and evaluation. The rule semantics are based on the Modbus protocol structure, especially request types and write related operations [8]. The prototype assumes that packet level Modbus traffic has already been parsed into structured events before rule evaluation takes place. This assumption is consistent with the parser first architecture discussed earlier and with existing industrial parsing approaches such as ICSNPP and related Modbus focused tooling [15, 16, 14].

For evaluation, the study primarily uses controlled normalized Modbus event sets created to represent specific detection scenarios. These datasets are intentionally small and focused because the purpose of the evaluation is not to create a large scale benchmark, but to verify

whether representative Modbus misuse scenarios can be expressed and detected through the proposed Sigma CPS approach. Public resources such as the CIC Modbus Dataset 2023 remain relevant as future external validation sources, but full parser to dataset integration is outside the present implementation scope [19].

The implementation is carried out in Python. Python is suitable for the prototype because it provides practical support for YAML processing, rule parsing, data transformation, and experimental evaluation. The prototype includes a rule loader, a validator, a translator, a normalizer, and an execution layer that applies translated detections to normalized Modbus events. The source code is organized into separate modules in order to keep the implementation aligned with the layered architecture defined later in the thesis.

3.4 Design and implementation process

The design and implementation process begins with defining a normalized event model for Modbus telemetry. This step is necessary because portable rules require stable field names and clear semantics. The normalized model includes the fields needed for the selected use cases, such as source host, destination host, Modbus function code, operation category, register address or range, and additional context required by the prototype.

After the event model is defined, the Sigma CPS schema is designed. The schema preserves the core Sigma idea of human readable YAML rules with metadata, detection selections, and conditions. At the same time, it extends the rule structure with Modbus aware fields so that detections can describe protocol specific behaviour in a structured form. The intention is to remain close enough to Sigma for readability and possible future compatibility, while still making the rule language expressive enough for Modbus specific use cases.

The next step is to implement the translator. The translator reads Sigma CPS Modbus rules, validates their structure, maps the rule content into an internal representation, and produces executable logic that can be applied to normalized telemetry. In the prototype, the output takes the form of Python based filtering and evaluation logic over structured events. The exact execution target is less important than demonstrating that the translation process is systematic and repeatable.

Once the translator is working, a small set of representative Modbus rules is created for

testing. These rules focus on detections that are both realistic and understandable, such as unauthorized write operations, writes to sensitive register ranges, exception generating write requests, and repeated write activity from a single source. These cases are suitable because they are clearly tied to Modbus semantics and can be explained without requiring a full process model of the monitored environment.

3.5 Evaluation criteria

The evaluation considers two dimensions: detection usefulness and engineering usefulness.

Detection usefulness refers to whether the proposed rules and prototype can identify the selected Modbus scenarios in a meaningful way. In this thesis, the evaluation does not rely on large scale statistical benchmarking. Instead, it verifies whether the implemented rules correctly match the intended normalized Modbus events and correctly reject events that should not match. Where relevant, the evaluation also considers whether the aggregation logic behaves as expected under repeated event conditions.

Engineering usefulness refers to whether the proposed Sigma CPS approach improves the representation of Modbus detection logic compared with ad hoc scripts or fully tool specific rules. This includes readability of the rule format, clarity of field definitions, ease of expressing representative detections, and the feasibility of translating the rules into executable logic.

The evaluation does not claim full coverage of Modbus threats or all CPS detection requirements. It is limited to a focused set of representative scenarios and to the prototype implementation developed in this thesis. The goal is to determine whether the proposed approach is feasible and useful as a rule abstraction layer, not to provide a complete operational detection framework.

Taken together, these evaluation criteria fit the scope of the thesis well. They allow the study to assess whether the proposed artifact works as intended, whether it remains understandable to human analysts, and whether it provides a meaningful proof of concept for portable Modbus detection engineering.

4 Design of the Sigma CPS Modbus Extension

This chapter presents the design of the proposed Sigma CPS Modbus extension. The purpose of the design is to define a Sigma-compatible rule model that can express Modbus specific detection logic in a readable, structured, and portable way. The chapter begins with the main design goals and requirements, then defines the normalized Modbus event model, explains how the Sigma schema is extended, specifies the rule structure and field vocabulary, and finally presents representative example rules.

4.1 Design goals and requirements

The design of Sigma CPS for Modbus is based on a simple idea: Modbus detection logic should be written once in a readable rule format and then translated into platform-specific detection logic after the telemetry has already been parsed into structured fields. This design direction follows the general Sigma model, where a rule contains metadata, log source information, and detection logic, while conversion and data-model adaptation are handled separately. In pySigma, this separation is reflected in processing pipelines that handle differences such as field naming and value representation before or during conversion [2, 3, 6].

The first design goal is compatibility with Sigma's core structure. The proposed extension should remain recognizably Sigma-like rather than introducing a completely new rule language. This is important for two reasons. First, a familiar rule structure lowers the learning curve for detection engineers who already understand Sigma. Second, staying close to the existing Sigma model makes future translation and tooling integration more realistic. For that reason, the Sigma CPS Modbus design keeps the standard high-level structure of rule metadata, log source description, and detection section, and only extends the model where Modbus specific semantics require additional expressiveness [2].

The second design goal is protocol awareness without raw-packet dependence. Modbus

detections often depend on protocol semantics such as function codes, addressing, request types, and operation intent. The official Modbus Application Protocol Specification defines these semantics explicitly, including the distinction between read and write functions and the addressing model used in requests [8]. However, writing portable detection rules directly against raw packet bytes would make the rules unreadable and tightly coupled to parser details. Therefore, Sigma CPS is designed to operate on normalized Modbus events rather than on raw network packets. In this model, protocol parsing happens first, and the rule layer consumes structured fields derived from the protocol.

The third design goal is backend independence. A rule should not be written specifically for one SIEM, one Zeek script, or one custom Python detector. Instead, the rule should describe detection intent in a platform-neutral way. This is one of the core strengths of Sigma and is also one of the main reasons for using a Sigma-compatible approach in the first place. The Modbus extension should therefore describe what to detect, not how a specific platform expresses that detection. Backend-specific syntax, field remapping, and output formatting belong to the translator layer rather than to the rule itself [2, 7].

The fourth design goal is readability for human analysts. A rule should remain understandable to a security engineer reading it as text. That means the rule should expose meaningful Modbus concepts such as operation type, function code, register range, unit identifier, source host, or destination host, rather than opaque internal implementation details. The thesis is not only about machine translation but also about making Modbus detection logic easier to review, share, and maintain. If the rule cannot be read and understood easily, then one of the central advantages of Sigma is lost.

The fifth design goal is support for representative Modbus misuse scenarios. The extension does not need to model every possible CPS detection pattern. Instead, it must be expressive enough for a focused set of practically meaningful Modbus detections. At minimum, the schema should support detections such as unauthorized write operations, writes to sensitive register ranges, requests originating from unauthorized sources, and repeated or excessive write activity. These use cases are appropriate because the Modbus standard clearly defines write-capable functions, and they also match the evaluation direction of this thesis [8, 9, 10].

The sixth design goal is compatibility with normalized telemetry pipelines. Operational technology environments differ from conventional enterprise IT because important security signals are often embedded in industrial communication semantics rather than only in standardized host logs. NIST highlights that OT environments have distinctive performance, reliability, and safety constraints, and this makes a parser-first architecture especially relevant in practice [1]. The Sigma CPS Modbus design therefore assumes a telemetry pipeline in which Modbus traffic is first decoded into structured events and only then evaluated by detection rules. This keeps the extension focused on rule representation and translation rather than on protocol decoding itself.

The seventh design goal is translator simplicity and implementability. Because this thesis includes a working prototype, the schema cannot be so abstract or complex that translation becomes unclear. Each rule element should map cleanly into an internal representation and then into executable logic. In practice, this means the schema should prefer explicit structured fields over ambiguous free-text constructs, and the condition model should remain close to standard Sigma semantics wherever possible.

Table 2 summarizes the main design goals.

Table 2. Design goals of the Sigma CPS Modbus extension.

Design goal	Meaning
Compatibility with Sigma core structure	Preserve the familiar Sigma rule layout and extend only where necessary.
Protocol awareness without raw-packet dependence	Represent Modbus semantics through normalized fields instead of raw packet bytes.
Backend independence	Keep the rule platform-neutral and leave target-specific mapping to the translator.
Readability for analysts	Use understandable Modbus concepts rather than implementation-specific details.
Support for representative misuse scenarios	Cover realistic Modbus detections such as unauthorized writes and sensitive register access.
Compatibility with normalized telemetry pipelines	Assume parser-generated structured telemetry before rule evaluation.
Translator simplicity and implementability	Keep the schema narrow enough for prototype parsing, validation, and execution.

Based on these goals, the main requirements of the design are as follows. First, the schema must preserve standard Sigma-style rule sections and only introduce Modbus specific extensions where necessary. Second, the rule model must operate on normalized Modbus fields rather than raw packet data. Third, the schema must represent core Modbus semantics relevant to detection, including function code, operation category, addressing information, and communication endpoints. Fourth, the rule must remain backend-neutral and translatable. Fifth, the rule syntax must remain readable to analysts. Sixth, the model must be narrow enough to support implementation and evaluation in a working prototype.

4.2 Normalized Modbus event model

The Sigma CPS Modbus rules in this thesis operate on normalized Modbus events rather than on raw packets. This is necessary because portable rules require stable field names and

consistent semantics across different monitoring environments. The thesis assumes that Modbus telemetry can be normalized into sufficiently stable fields to support rule portability and defines the prototype around structured events produced before rule evaluation.

In this design, one normalized event represents one Modbus request, with optional response information attached when it is available. This choice keeps the model simple and fits the main detection scenarios in the thesis, since most security-relevant Modbus behaviour is expressed by client requests such as reads, writes, scans, and repeated access patterns. Response-related information is still included where useful, especially for exception handling and validation.

The event model is divided into five parts: general metadata, network context, Modbus protocol fields, derived semantic fields, and optional response fields. Figure 1 illustrates these groups and their constituent fields. Table 3 then presents the full field definitions used by the prototype.

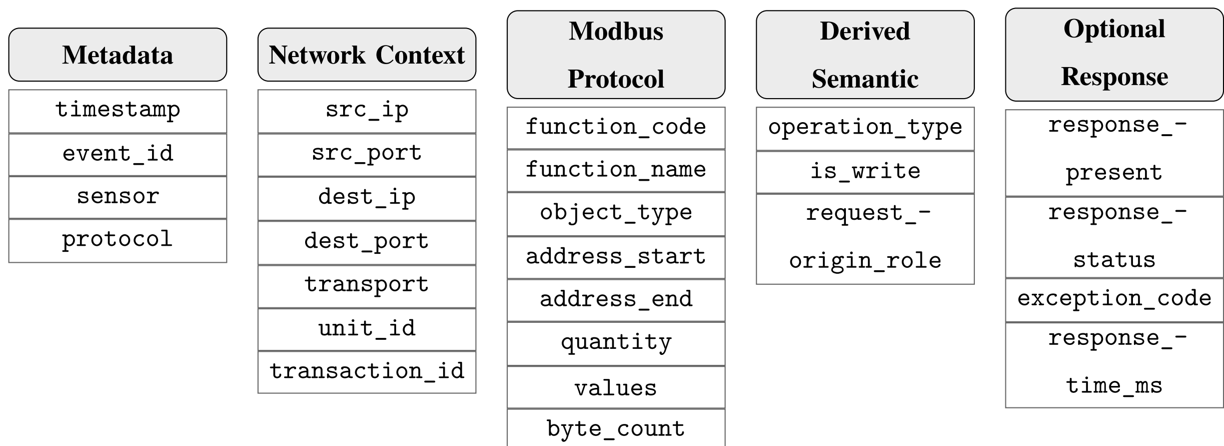

Figure 1. Normalized Modbus event model grouped by field category.

Table 3. Core normalized Modbus event fields used by the prototype.

Field	Type	Description
timestamp	datetime	Time at which the request was observed.
event_id	string	Optional unique identifier for the event.
sensor	string	Optional identifier of the sensor or collector.
protocol	string	Protocol label, defaulting to modbus_tcp.
src_ip	string	Source IP address of the request originator.
src_port	integer	Source TCP port of the request.
dest_ip	string	Destination IP address.
dest_port	integer	Destination TCP port, defaulting to 502.
transport	string	Transport layer protocol, defaulting to tcp.
unit_id	integer	Modbus unit identifier.
transaction_id	integer	Modbus/TCP transaction identifier, if available.
function_code	integer	Numeric Modbus function code.
function_name	string	Human-readable function label derived from the function code.
object_type	string	Target type such as coil, holding_register, or input_register.
address_start	integer	First register or coil address referenced by the request.
address_end	integer	Derived ending address based on the starting address and quantity.
quantity	integer	Number of addressed items.
values	list/string	Values written by the request, where applicable.
byte_count	integer	Byte count field from the Modbus PDU, if present.
operation_type	string	High-level category such as read, write, diagnostic, or other. Derived from function code if not supplied.
is_write	boolean	Indicates whether the request is write-capable. Derived from function code if not supplied.
request_origin_role	string	Optional enrichment field such as hmi, plc, or engineering_station.
response_present	boolean	Indicates whether a response was observed for this request.
response_status	string	Optional normalized response result such as success or exception.
exception_code	integer	Modbus exception code, if present.
response_time_ms	float	Round-trip response time in milliseconds, if available.

Several design decisions follow from this model. First, the rules are written against semantic fields, not parser-specific names. For example, a Sigma CPS rule should refer to `function_code` or `operation_type`, not to whatever raw label a specific tool uses internally. This keeps the rule portable and shifts parser-specific mapping into the normalization or translation layer.

Second, the model includes both raw protocol fields and derived fields. The raw fields preserve fidelity to Modbus semantics, while the derived fields improve readability and simplify rule authoring. A detection engineer may choose to match on `function_code: 16` but may also prefer a more readable condition such as `operation_type: write`. Supporting both levels makes the rule language more practical without losing precision.

Third, the model is designed to support the specific Modbus scenarios used later in the thesis. Unauthorized write detections require fields such as source IP address, unit identifier, function code, and address start. Sensitive register monitoring additionally needs address-end or address-range logic. Repeated or abnormal write behaviour requires timestamped events with operation categories. Exception-based detections require optional response metadata.

In summary, the normalized Modbus event model provides the stable field layer on which the Sigma CPS extension is built. It separates protocol parsing from rule authoring, preserves the Modbus concepts needed for detection, and keeps the schema narrow enough for prototype implementation.

4.3 Extension of the Sigma schema for Modbus telemetry

The proposed Sigma CPS Modbus schema keeps the basic structure of a standard Sigma rule but extends the rule content so that it can describe Modbus specific telemetry. The purpose of the extension is not to replace Sigma with a new language. Instead, it adds a Modbus-aware layer on top of the existing rule model so that protocol-level detections can be written in a readable and portable format. This design follows the thesis objective of preserving readability and portability while making CPS detection logic translatable into executable platform-specific detections.

At a high level, a Sigma CPS Modbus rule still contains the usual components: rule

metadata, log source description, detection logic, and supporting attributes such as level, tags, and status. The extension is introduced in two places. First, the `logsource` section is specialized so that the rule clearly declares that it targets normalized Modbus telemetry. Second, the `detection` section is allowed to reference Modbus-aware fields from the normalized event model defined in the previous section.

The first extension concerns the log source declaration. In a conventional Sigma rule, `logsource` identifies the category, product, and service associated with the events being matched. For Sigma CPS Modbus rules, this section states that the rule applies to Modbus telemetry rather than to a conventional operating-system or application log source. A typical structure therefore uses values such as `category: network`, `product: ot`, and `service: modbus`. The important point is that the rule explicitly identifies the telemetry domain so that translators and validation logic can interpret the rule correctly.

The second extension concerns the field vocabulary available inside the detection logic. Standard Sigma rules usually assume stable log fields such as process names, file paths, command lines, or event identifiers. These do not reflect the semantics of Modbus communication. Sigma CPS therefore allows the detection section to reference normalized Modbus fields such as `function_code`, `operation_type`, `unit_id`, `address_start`, `address_end`, `quantity`, `src_ip`, `dest_ip`, and `response_status`. These fields are not arbitrary additions. They are the rule-level representation of the normalized Modbus event model and support the representative Modbus detection scenarios identified earlier in the thesis.

The third extension is the introduction of semantic convenience fields. Some Modbus detections are easier to express through derived meanings than through raw numeric values alone. For example, matching on `function_code: 6` is precise, but matching on `operation_type: write` is easier to read and explain. Sigma CPS therefore permits both kinds of fields. Numeric protocol details remain available when precision is needed, while semantic fields improve readability and simplify rule authoring.

The fourth extension is support for address-based conditions. In many Modbus use cases, the security meaning of a request depends not only on the fact that it is a write operation, but also on which registers or coils are being accessed. For this reason, the schema must

support matching on exact addresses, address ranges, and derived end addresses. This makes it possible to represent detections such as writes to sensitive register blocks or requests that touch reserved address ranges.

The fifth extension is support for communication-context conditions. Modbus detections frequently depend on who sent the request and where it was sent. A write operation originating from an expected HMI may be normal, while the same operation from an engineering workstation or unknown host may be suspicious. Sigma CPS therefore allows endpoint context such as source and destination identifiers to be part of the detection logic. In the prototype, this is handled through normalized fields such as `src_ip`, `dest_ip`, and optional enrichment fields such as `request_origin_role` when available.

The sixth extension is support for lightweight behavioural conditions. The thesis does not attempt to build a full process-aware anomaly-detection language, but it does need to represent simple repeated or unusual Modbus behaviours such as frequent write requests from a single source or repeated access to a sensitive target. Where possible, Sigma CPS remains close to Sigma's condition style and uses repeated-event logic only in a limited and controlled way. This keeps the prototype feasible while still allowing the thesis to demonstrate that Modbus detection is not limited to single-event matches [21].

4.4 Rule structure and field definitions

This section defines the internal structure of a Sigma CPS Modbus rule and the field vocabulary supported by the proposed schema. The purpose of this definition is to make the design precise enough for implementation. In other words, this section turns the design goals and the normalized event model into a concrete rule model that can be parsed, validated, and translated by the prototype.

A Sigma CPS Modbus rule follows the general organization of a Sigma rule, but the allowed field vocabulary inside the detection logic is specialized for normalized Modbus telemetry. The rule consists of six main parts: metadata, log source, detection selections, condition logic, optional behavioural constraints, and severity or classification fields.

The metadata section identifies and documents the rule. Most of these fields remain unchanged from ordinary Sigma usage because portability and readability depend on rule

metadata as much as on field-level logic. Typical metadata includes the rule title, identifier, status, description, author, date, references, tags, false-positive notes, and severity level.

The `logsource` section declares the type of telemetry the rule expects. For this thesis, the rule source is normalized Modbus telemetry. The most important function of `logsource` is to ensure that the translator knows which field vocabulary and backend mappings apply to the rule.

The `detection` section is the core of the schema. It contains named selections and a final condition statement combining those selections. This remains close to Sigma's original design because named selections are both readable and easy to translate. Each selection contains one or more field constraints. Table 4 summarizes the operator set supported by the prototype.

Table 4. Supported field operators in the prototype.

Operator	**Purpose**	**Example**
exact match	Match an exact value	`operation_type: write`
`in`	Match one of several values	`function_code|in: [5,6,15,16]`
`not_in`	Exclude listed values	`src_ip|not_in: [192.168.1.10]`
`gte`	Greater than or equal	`address_start|gte: 40001`
`lte`	Less than or equal	`address_end|lte: 40010`
`gt`	Greater than	`quantity|gt: 10`
`lt`	Less than	`quantity|lt: 3`
`contains`	String or list containment	`values|contains: 9999`
`startswith`	Prefix matching for strings	`function_-name|startswith: Write`
`endswith`	Suffix matching for strings	`request_origin_-role|endswith: station`

To support lightweight behavioural detections without introducing a full anomaly-detection

language, the schema allows two optional fields inside `detection`: `timeframe` and `aggregation`. The `timeframe` field defines the observation window, while `aggregation` specifies grouping and count-threshold logic. This is intended for simple use cases such as repeated writes from one source in a short period.

The prototype also applies basic validation rules. Every rule must contain a title, identifier, `logsource`, `detection`, and condition. The `logsource.service` field must identify Modbus for Modbus specific rules. Every field referenced in `detection` must belong to the allowed normalized Modbus field vocabulary. Aggregation cannot be used without a timeframe. Range conditions must use numeric fields such as `address_start`, `address_end`, or `quantity`. Finally, the translator rejects rules that use unsupported operators or undefined selections.

In summary, the rule structure and field definitions provide the formal layer needed for Sigma CPS Modbus rules. The schema remains recognizably Sigma-like, but its detection vocabulary is specialized for normalized Modbus telemetry and for the representative use cases selected in this thesis.

4.5 Example Sigma CPS Modbus rules

This section presents example Sigma CPS Modbus rules that illustrate how the proposed schema can be used in practice. The purpose of these examples is not to claim full coverage of Modbus threats, but to demonstrate that the designed rule model can express representative misuse scenarios in a readable and structured way.

The first example detects write-capable Modbus requests originating from a source that is not part of an approved allowlist.

Listing 4.1. Example Sigma CPS Modbus rule for unauthorized write requests.

```
title: Unauthorized Modbus Write from Non-Approved Source
id: 1a4b2f7e-0001-4c9b-a111-000000000001
status: experimental
description: Detects Modbus write requests from sources outside the
    approved host list.
author: Adeyimika Adetunji
date: 2026-01-10
```

```
references:
  - MODBUS Application Protocol Specification V1.1b3
tags:
  - ot.modbus
  - attack.unauthorized-command
logsource:
  category: network
  product: ot
  service: modbus
detection:
  selection_write:
    operation_type: write
  selection_source:
    src_ip|not_in:
      - 192.168.1.10
      - 192.168.1.11
  condition: selection_write and selection_source
level: high
```

This rule shows the basic advantage of the proposed design. The detection is expressed in terms of readable Modbus-aware semantics such as `operation_type` and `src_ip`, while the translator remains responsible for turning that logic into the syntax required by the target platform.

The second example focuses on address-based detection by identifying write operations targeting a protected register range.

Listing 4.2. Example Sigma CPS Modbus rule for sensitive register ranges.

```
title: Modbus Write to Sensitive Register Range
id: 1a4b2f7e-0002-4c9b-a111-000000000002
status: experimental
description: Detects write operations targeting a protected
   register range.
author: Adeyimika Adetunji
date: 2026-01-10
references:
  - MODBUS Application Protocol Specification V1.1b3
tags:
```

```
  - ot.modbus
  - attack.impact
logsource:
  category: network
  product: ot
  service: modbus
detection:
  selection_write:
    operation_type: write
    object_type: holding_register
  selection_range:
    address_start|gte: 40001
    address_end|lte: 40010
  condition: selection_write and selection_range
level: high
```

The third example adds lightweight behavioural logic. It detects repeated write activity from the same source, destination, and unit identifier within a short time window.

Listing 4.3. Example Sigma CPS Modbus aggregation rule for repeated write activity.

```
title: Excessive Modbus Write Activity from Single Source
id: 1a4b2f7e-0005-4c9b-a111-000000000005
status: experimental
description: Detects repeated Modbus write activity from one source
    within a short
time window.
author: Adeyimika Adetunji
date: 2026-01-10
references:
  - MODBUS Application Protocol Specification V1.1b3
tags:
  - ot.modbus
  - attack.impact
logsource:
  category: network
  product: ot
  service: modbus
detection:
  selection_write:
```

```
    operation_type: write
  timeframe: 1m
  aggregation:
    group_by:
      - src_ip
      - dest_ip
      - unit_id
    count: 3
  condition: selection_write
level: medium
```

Taken together, these examples show that the proposed schema can represent several important classes of Modbus detection logic in a consistent way. The rules remain structurally similar to ordinary Sigma rules, but the selectable fields and conditions are specialized for normalized Modbus telemetry. This supports the central thesis claim that Modbus detections can be authored once in a Sigma-compatible format and later translated into platform-specific executable logic.

The examples also highlight the balance the design tries to achieve. On one hand, the schema includes enough protocol-specific detail to express realistic Modbus detections. On the other hand, it remains narrow enough to support parsing, validation, and translation in a prototype implementation. This balance is important because the thesis contribution is not a complete industrial monitoring framework, but a focused proof of concept showing that portable Modbus detection engineering is feasible within a Sigma-compatible model.

5 Prototype Implementation

This chapter describes the implementation of the proposed Sigma CPS Modbus prototype. While the previous chapter defined the design goals, event model, schema extensions, and example rules, the purpose of this chapter is to show how those ideas were realized in a working system. In line with the scope of the thesis, the implementation focuses on Modbus as the selected protocol and demonstrates how Sigma compatible rules can be parsed, validated, translated, and executed over normalized Modbus telemetry.

The prototype is not intended to be a production ready detection platform. Its purpose is to demonstrate the feasibility of the proposed approach. More specifically, it shows that Modbus detection logic can be written in a Sigma compatible rule format and systematically translated into executable logic without rewriting the detection separately for each target environment. This directly supports the main thesis goal of improving the portability and readability of CPS detection rules.

5.1 System architecture

The prototype follows a layered architecture. This architecture was chosen because the thesis does not aim to replace protocol parsers or network monitoring tools. Instead, it introduces a rule abstraction and translation layer above normalized Modbus telemetry. The implementation therefore assumes that raw Modbus traffic has already been parsed into structured events before the Sigma CPS rules are applied.

The system consists of five main components:

1. **Telemetry input layer.** This layer ingests normalized Modbus events. The input may come from preprocessed log files, parsed packet captures, or synthetic test data created for evaluation. The only strict requirement is that the input conforms to the normalized event model defined in Chapter 4.
2. **Rule input layer.** This layer reads Sigma CPS Modbus rules written in YAML

format. These rules contain metadata, a Modbus aware `logsource`, detection selections, optional behavioural constraints, and a final condition statement.

3. **Parser and validator.** The parser loads the YAML rule and converts it into an internal Python representation. The validator checks whether the rule is structurally correct, whether all referenced fields belong to the allowed Modbus vocabulary, and whether unsupported operators or invalid combinations are present.
4. **Translator.** The translator converts the validated Sigma CPS rule into executable detection logic. In the prototype, this logic is represented as structured filtering and evaluation logic over normalized Modbus events. The translator is responsible for interpreting selections, operators, conditions, and optional aggregation constraints.
5. **Execution and result layer.** This layer applies the translated rule logic to the Modbus event stream or dataset and produces detection results. The results may include matched events, rule hit counts, and short explanations of why a match occurred.

The complete prototype source code, rule files, normalized event samples, selected outputs, and automated tests are available in the accompanying GitHub repository [25].

Figure 2 illustrates the overall system architecture and the flow of data through the prototype pipeline.

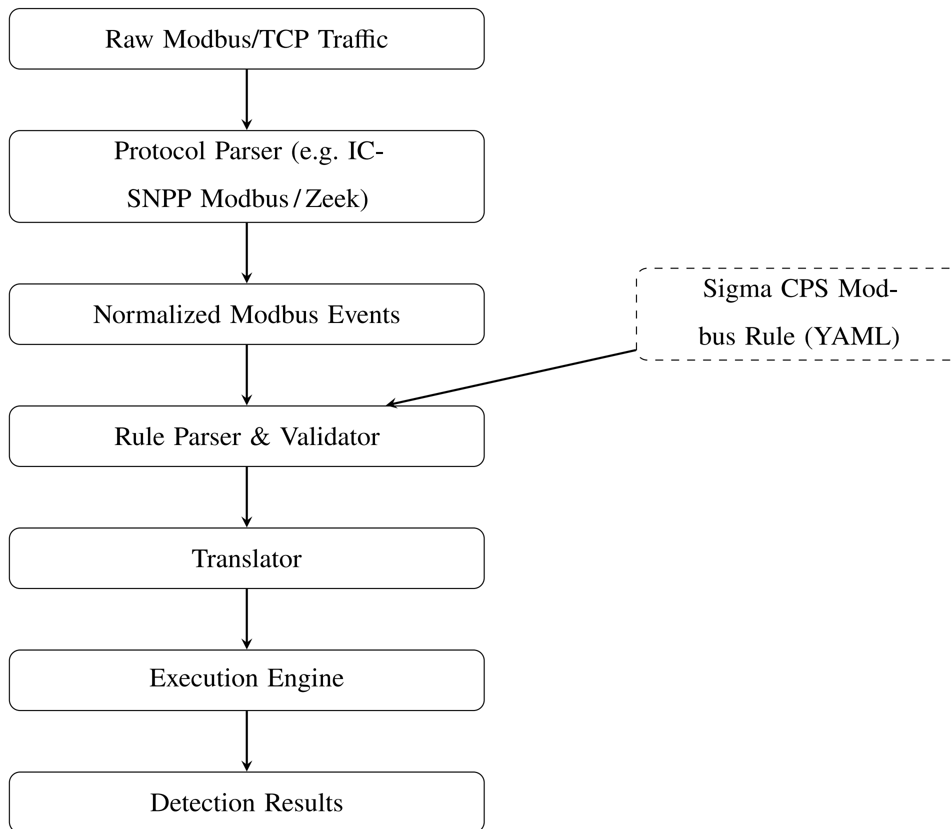

Figure 2. Overall Sigma CPS Modbus prototype architecture.

Figure 3 shows the directory structure of the Sigma CPS Modbus prototype, including the rule files, normalized datasets, source modules, and automated tests.

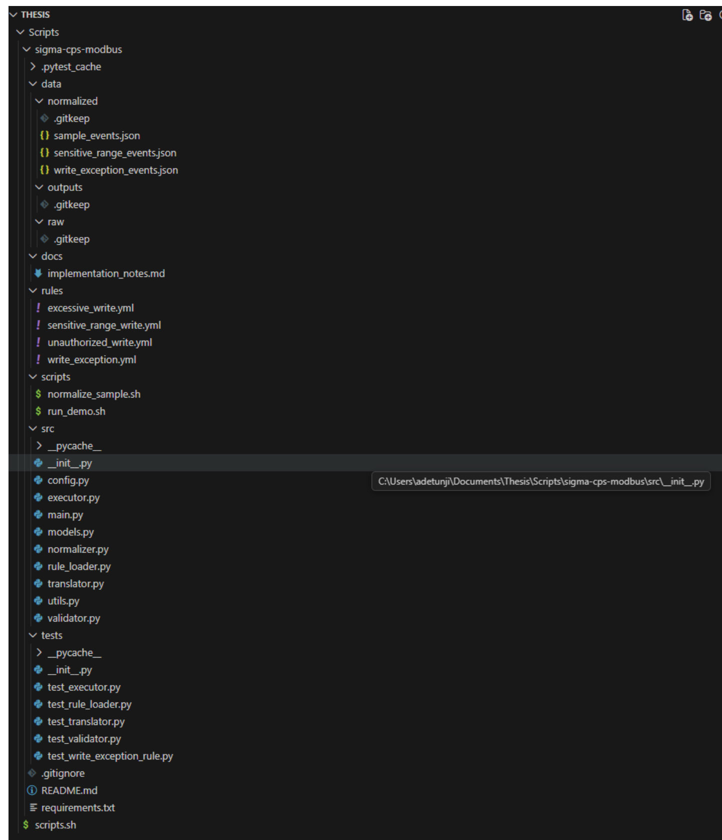

Figure 3. Directory structure of the Sigma CPS Modbus prototype.

This structure reflects the central idea of the thesis: write the rule once in a portable form, then translate and execute it against structured Modbus telemetry.

A key implementation decision is that the prototype separates rule meaning from execution logic. The YAML rule expresses what should be detected, while the translated internal representation determines how the detection is evaluated in the prototype. This separation is important because it mirrors the same portability principle that motivated Sigma in the first place.

5.2 Input and output design

The prototype uses two primary inputs: normalized Modbus telemetry and Sigma CPS Modbus rules.

The first input is a structured Modbus event dataset. Each event contains the fields required by the prototype, such as timestamp, source and destination information, function code,

operation type, address range, and optional response information. These events may be stored in JSON or Python native structures, as long as the field names remain consistent with the normalized event model.

The second input is a Sigma CPS rule file in YAML format. This file contains the metadata and detection logic defined in the design chapter. Each rule is treated as a self contained detection specification.

The output of the prototype consists of translated rule objects and detection results. A translated rule object is the internal representation produced after parsing and validation. Detection results include the events that matched the rule and additional details such as which rule triggered and why the event satisfied the rule condition. For aggregation based rules, the output may also include grouped counts and the corresponding observation window.

This input/output model keeps the system simple and makes evaluation straightforward. It also supports reproducibility, since the same normalized data and rule file can be re executed without changing the prototype logic.

Examples of the normalized event files and representative Sigma CPS Modbus rule files used by the prototype are provided in Appendix 3.

5.3 Rule parser and validator

The first core implementation component of the prototype is the rule parser and validator. Its purpose is to load Sigma CPS Modbus rules written in YAML, convert them into an internal representation, and reject rules that do not comply with the schema defined in Chapter 4. This component is necessary because the prototype does not execute raw YAML directly. Instead, each rule is transformed into a structured Python object that can be interpreted consistently by the translator and execution engine.

The parser reads the YAML rule file and extracts its top level sections, including metadata, `logsource`, and `detection`. Named selections inside the `detection` block are converted into structured field constraints. Each constraint is represented internally by a field name, an operator, and a value. This makes the later translation step systematic, since the translator

operates on a normalized internal rule model rather than on raw YAML text.

The validator enforces the constraints of the Sigma CPS Modbus schema. It checks that required top level fields are present, that the rule targets Modbus telemetry, that only supported Modbus fields are used, and that all referenced operators are valid. It also validates the condition expression by ensuring that every named selection referenced in the condition exists in the rule. For aggregation based detections, the validator ensures that a timeframe is provided and that grouping fields belong to the allowed field vocabulary.

This implementation choice improves robustness in two ways. First, it prevents malformed rules from reaching the translation layer. Second, it makes rule behaviour more predictable because unsupported constructs are rejected explicitly rather than failing silently at runtime. In practical terms, this means that the prototype behaves like a controlled rule processing system rather than a loose collection of parsing scripts.

5.4 Translation workflow

After a rule has been loaded and validated, it is passed to the translator. The purpose of the translator is to convert the Sigma CPS Modbus rule into executable Python based detection logic over normalized Modbus events. This translation step is the central proof of concept contribution of the prototype, because it demonstrates that a Sigma compatible Modbus rule can be transformed into a working backend representation instead of being written directly in a platform specific query language.

Figure 4 illustrates the translation workflow from YAML rule input to executable predicate output.

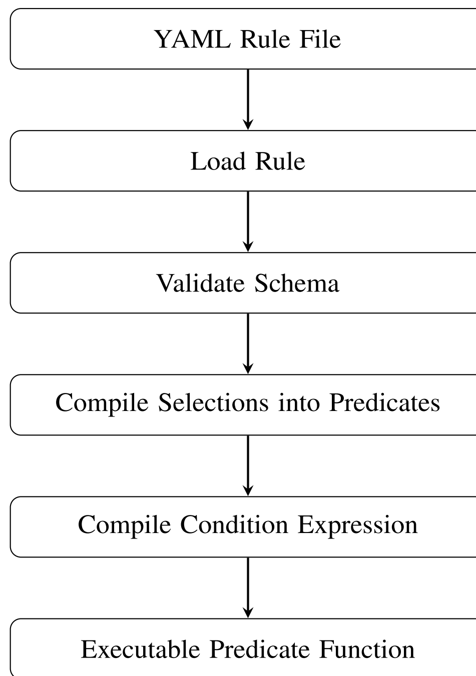

Figure 4. Rule translation workflow from YAML input to executable predicate.

The translator processes the rule in three stages. First, it compiles each field constraint into a predicate function. For example, an exact match such as `operation_type: write` becomes a predicate that compares the event field against the expected value. Similarly, operators such as `not_in`, `gte`, and `lte` are mapped into corresponding Python comparison logic.

Second, the translator compiles each named selection into a selection level predicate. A selection is satisfied when all of its constraints match the current event. This means that selections preserve the same logical role they have in Sigma: each one represents a meaningful detection fragment, and the final rule condition combines these fragments.

Third, the translator parses the rule condition and compiles it into a final event matching function. In the current prototype, the supported condition logic includes simple Boolean combinations such as `and`, `or`, and `not`. This is sufficient for the selected Modbus use cases and keeps the implementation narrow enough to remain feasible within the thesis scope.

For aggregation based rules, the translator does not implement a separate query language. Instead, the rule carries the additional `timeframe` and `aggregation` parameters, and the execution layer interprets these parameters during runtime. This design keeps the

translation model simple while still allowing the prototype to support repeated event detections.

5.5 Execution over normalized Modbus telemetry

The execution layer applies translated rules to normalized Modbus events. The prototype supports two execution modes: single event matching and aggregation based matching.

In single event mode, the execution engine evaluates each event independently. If the translated rule condition returns true for a given event, the engine records a match result containing the rule identifier, rule title, matched event, and a short reason string. This mode is used for detections such as unauthorized write operations, writes to sensitive register ranges, and write requests resulting in exception responses.

In aggregation mode, the execution engine evaluates repeated behaviour over a time window. Figure 5 illustrates the aggregation workflow used for repeated event detections.

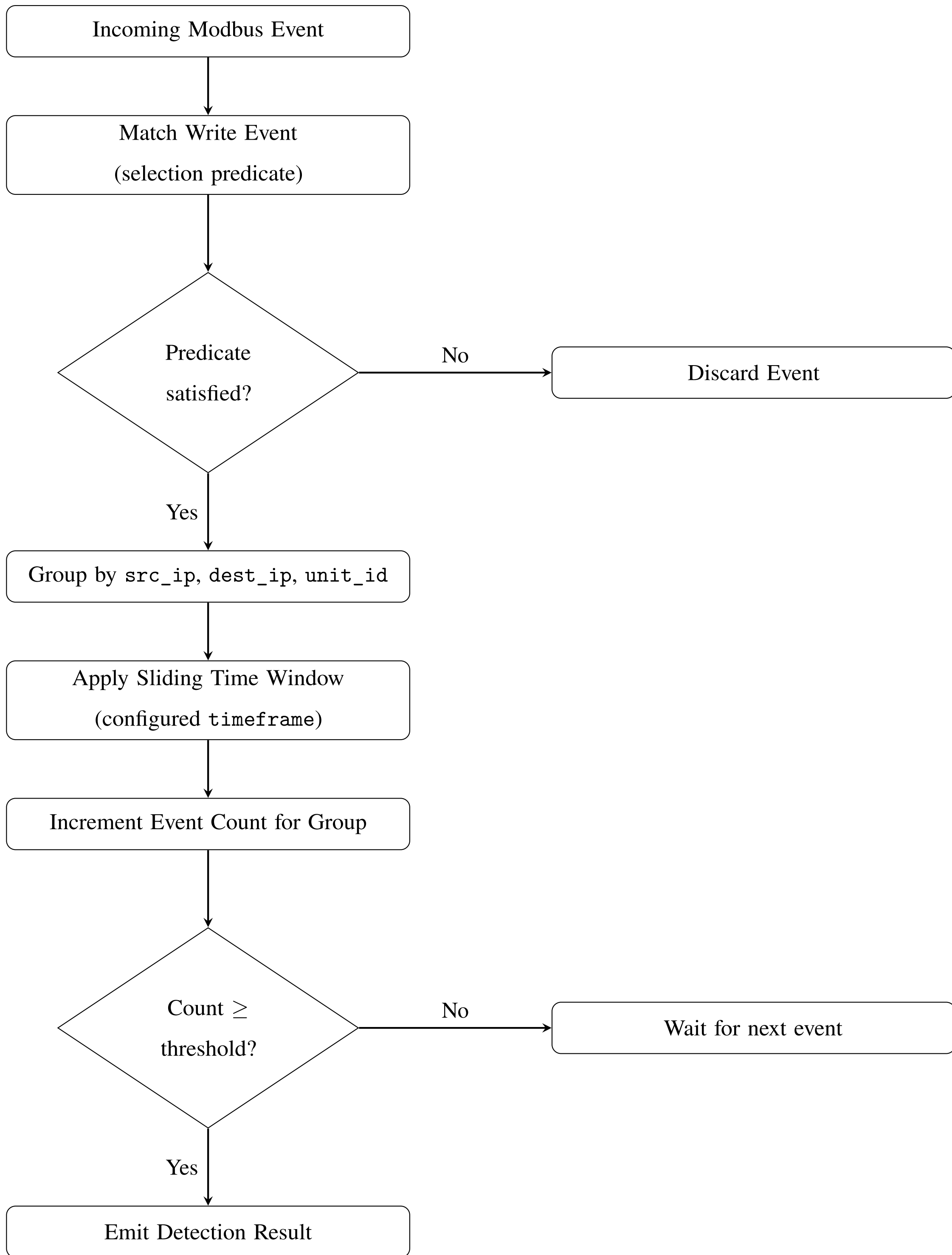

Figure 5. Aggregation workflow for repeated event detection.

The current prototype supports a limited but useful aggregation model consisting of a timeframe, grouping fields, and a count threshold. Matching events are first grouped according to the specified fields, such as source IP address, destination IP address, and unit identifier. For each group, the engine maintains a sliding time window and counts

how many matching events occur inside that window. When the threshold is reached or exceeded, the engine records a match result.

This approach was tested successfully with a rule detecting excessive Modbus write activity from a single source. In that scenario, the threshold was configured as three write events within one minute for the same source, destination, and unit identifier. The prototype correctly generated a detection when the third matching event appeared in the window. This confirms that the schema and execution model can support both ordinary point detections and limited behavioural detections in a consistent way.

The implementation also includes a normalization step before execution. Although the sample data used in the prototype was already close to the intended structure, the normalizer derives fields such as `operation_type`, `object_type`, `is_write`, and `address_end` from rawer Modbus related attributes. This supports the architectural claim made earlier in the thesis that rule evaluation should happen over normalized Modbus semantics rather than over raw packet data or parser specific field names.

Table 5 summarizes the core function code mapping used directly by the prototype implementation. Function names follow the MODBUS Application Protocol Specification, while the derived fields `operation_type`, `object_type`, and `is_write` are author defined normalization fields derived from those protocol semantics [8]. A fuller mapping, including additional function codes and exception handling details, is provided in Appendix 2.

Table 5. Core function code mapping used in the prototype implementation.

Code	Function name	operation_-type	object_type	is_write
1	Read Coils	read	coil	false
3	Read Holding Registers	read	holding_register	false
5	Write Single Coil	write	coil	true
6	Write Single Register	write	holding_register	true
15	Write Multiple Coils	write	coil	true
16	Write Multiple Registers	write	holding_register	true
22	Mask Write Register	write	holding_register	true
23	Read/Write Multiple Registers	read_write	holding_register	true

The mapping in Table 5 is sufficient for the rule types implemented in the prototype, while Appendix 2 provides the extended normalization mapping and related protocol details.

To illustrate the kind of protocol level evidence on which the normalization layer is based, Figure 6 shows an example of Modbus/TCP traffic observed in Wireshark. Figures 6 and 7 are author created Wireshark screenshots based on packet captures from the ICS_-PCAPS MODBUS/TCP#1 dataset release [26]. The release describes the captures as MODBUS/TCP traffic generated in a small scale CPS/SCADA testbed and distributed as PCAP files for research use.

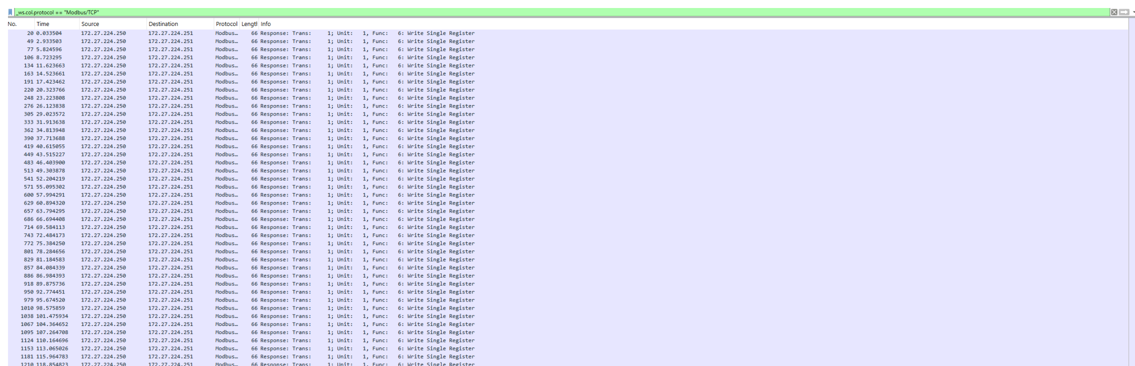

Figure 6. Example of Modbus/TCP traffic observed in Wireshark. Source: author's screenshot based on ICS_PCAPS MODBUS/TCP#1 packet captures [26].

Figure 7 shows the detailed decode of a Modbus/TCP packet, including the transaction identifier, unit identifier, function code, reference number, and register value that motivate the normalized fields used by the prototype.

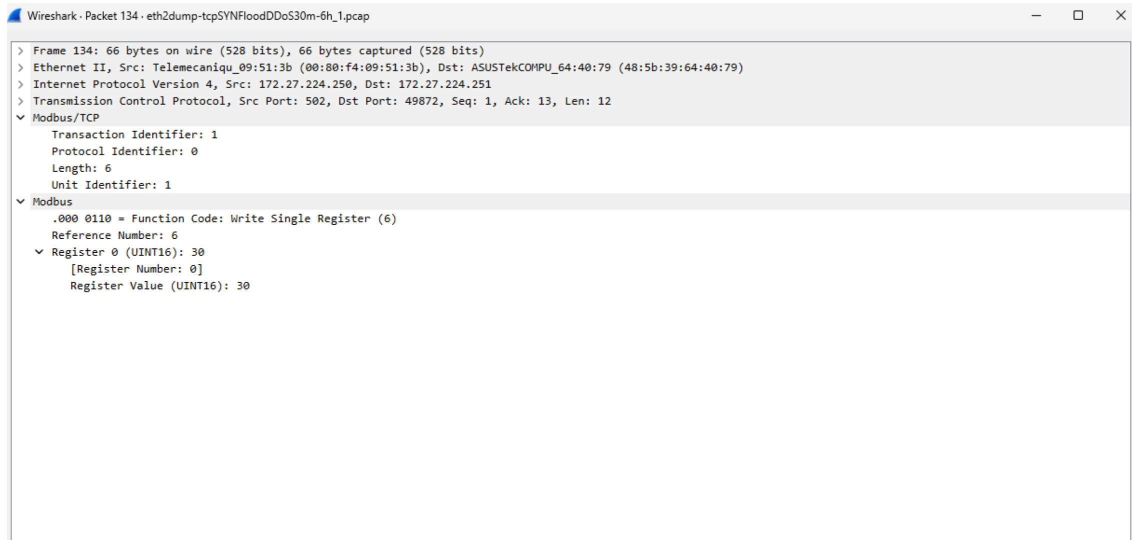

Figure 7. Detailed Modbus packet decode in Wireshark showing transaction level and function level fields. Source: author's screenshot based on ICS_PCAPS MODBUS/TCP#1 packet captures [26].

5.6 Implemented rules and observed prototype behaviour

At the current stage of implementation, the prototype supports four representative Modbus rule types.

The first rule detects write capable requests from non approved source hosts. When executed over the prepared sample data, this rule correctly matched two events whose source IP addresses were outside the configured allowlist.

The second rule detects writes to a protected register range. In the corresponding test dataset, the prototype correctly matched one event whose write operation targeted the protected range and rejected the events that either fell outside the range or used a read operation instead of a write.

The third rule detects Modbus write requests that result in exception responses. In the prepared sample data, the prototype correctly matched the single event that combined a write operation with an exception response and rejected the events that only partially satisfied the rule logic.

The fourth rule detects excessive Modbus write activity from a single source. This rule used the lightweight aggregation extension of the schema. The prototype correctly generated a match when three write events from the same source, to the same destination and unit identifier, occurred within the configured one minute window.

These results show that the prototype already supports the main rule categories needed for the thesis: source based restrictions, address based restrictions, response based detections, and simple repeated event detections. This is important because it demonstrates that the proposed Sigma CPS Modbus model is not only theoretically expressive but also executable in practice.

Table 6 summarizes the main implementation modules of the prototype.

Table 6. Prototype modules and responsibilities.

Module	**Responsibility**
`rule_loader.py`	Loads YAML Sigma CPS rules and converts them into an internal rule representation.
`validator.py`	Validates schema structure, allowed fields, operators, and condition expressions.
`translator.py`	Converts validated rules into executable Python based predicates.
`executor.py`	Executes translated logic over normalized Modbus events, including aggregation.
`normalizer.py`	Derives normalized Modbus fields from structured input data.
`main.py`	Provides a runnable entry point for demonstration and testing.

Full example rule files, selected normalized event samples, and representative execution outputs are included in Appendix 3.

5.7 Prototype limitations

Although the prototype demonstrates feasibility, it has several limitations. First, the execution target is Python based detection logic rather than a production SIEM backend. This was a deliberate design decision made to keep the prototype controllable and implementable within the thesis scope.

Second, the current condition model is intentionally limited to simple Boolean combinations of selections. More advanced Sigma style constructs, correlation rules, and complex temporal logic are outside the present implementation.

Third, the aggregation support is narrow. It is sufficient for repeated write demonstrations, but it is not intended to replace stream processing systems or full correlation engines.

Fourth, the prototype assumes the availability of normalized Modbus events. It does not include a full parser from raw packet captures into the final event model. Instead, it implements the normalization layer above already structured Modbus related input.

These limitations do not weaken the main contribution of the thesis. On the contrary, they help define it clearly: the work is a focused proof of concept showing that Modbus detection logic can be represented in a Sigma compatible rule model and translated into executable logic through a compact prototype.

6 Evaluation and Results

This chapter evaluates the implemented Sigma CPS Modbus prototype and presents the observed results. The goal of the evaluation is not to provide a large scale industrial benchmark, but to determine whether the prototype can correctly process Sigma CPS Modbus rules and execute them over normalized Modbus telemetry in a consistent and repeatable way.

6.1 Evaluation setup

The purpose of the evaluation was to determine whether the prototype could correctly process Sigma CPS Modbus rules and execute them over normalized Modbus telemetry. In line with the scope of the thesis, the evaluation was not designed as a large scale industrial benchmark. Instead, it focused on a small number of representative Modbus detection scenarios that reflect the use cases defined earlier in the thesis: unauthorized write operations, writes to sensitive register ranges, write requests resulting in exceptions, and repeated write activity from a single source. This evaluation approach is consistent with the narrowed Modbus focused implementation and with the thesis goal of demonstrating feasibility through a working prototype rather than building a complete production detection platform.

The evaluated prototype consisted of the following implemented components: a YAML rule loader, a schema validator, a translator that converts Sigma CPS Modbus rules into executable Python logic, a normalizer for Modbus event fields, and an execution engine capable of both single event and aggregation based matching. The input data for the evaluation consisted of normalized Modbus event files created for each scenario. These files were intentionally small and controlled so that rule behaviour could be verified precisely.

The evaluation was performed in two complementary ways. First, each rule was executed through the prototype pipeline using sample event data. Second, automated tests were used

to verify expected behaviour in the loader, validator, translator, and executor modules. This combination allowed both functional demonstration and repeatable verification.

It is important to state clearly what this evaluation does and does not cover. The present implementation proves that Sigma CPS Modbus rules can be parsed, validated, translated, and executed correctly in a prototype backend. It does not yet evaluate the approach against a large public dataset or through a full production SIEM integration. That limitation is deliberate and follows from the thesis scope: the main research contribution lies in the rule representation and translation layer, not in packet parsing or end to end operational deployment.

6.2 Detection scenarios

Four representative rule scenarios were used in the evaluation.

The first scenario tested detection of unauthorized Modbus write requests from non approved source hosts. The rule matched write operations where the source IP address was not part of a configured allowlist.

The second scenario tested detection of write operations targeting a protected register range. This rule required both a write type operation and an address span within a predefined sensitive range.

The third scenario tested detection of Modbus write requests that resulted in exception responses. This scenario evaluated whether the prototype could combine request semantics with response metadata.

The fourth scenario tested repeated write activity from a single source within a short time window. This scenario used the lightweight aggregation model of the schema, where matching events were grouped by source IP address, destination IP address, and unit identifier, and then counted within a one minute window.

These scenarios were selected because together they cover the main kinds of behaviour the prototype was designed to support: source based restrictions, address based restrictions, response based detections, and repeated event detections.

Table 7 summarizes the evaluation scenarios and expected outcomes.

Table 7. Evaluation scenarios and expected outcomes.

Scenario	**Expected outcome**	**Expected matches**
Unauthorized write from non approved source	Match all write events whose source IP is outside the configured allowlist.	2
Write to sensitive register range	Match only write events that target the protected register interval.	1
Write request resulting in exception	Match only events that are both write operations and exception responses.	1
Excessive write activity from single source	Match when the aggregation threshold is reached within the configured one minute window.	1

6.3 Results

Table 8 summarizes the observed rule execution results of the prototype.

Table 8. Summary of prototype rule execution results.

Rule	**Detection type**	**Expected outcome**	**Observed result**
Unauthorized Modbus Write from Non Approved Source	Single event	Match all write events from non allowlisted sources	2 matches
Modbus Write to Sensitive Register Range	Single event	Match write events targeting protected range only	1 match
Modbus Write Request Resulting in Exception	Single event	Match write events with exception response only	1 match
Excessive Modbus Write Activity from Single Source	Aggregation	Match when threshold is reached within one minute	1 match

Figure 8 presents an example of the prototype running the unauthorized Modbus write rule. The figure shows the Sigma CPS rule definition, the normalized Modbus input events, and

the resulting matches produced by the execution engine.

```
Scripts > sigma-cps-modbus > rules > ! unauthorized_write.yml
1  title: Unauthorized Modbus Write from Non-Approved Source
2  id: 1a4b2f7e-0001-4c9b-a111-000000000001
3  status: experimental
4  description: Detects Modbus write requests from sources outside
5  author: Adeyimika Adetunji
6  date: 2026-01-10
7  references:
8    - Modbus Application Protocol Specification V1.1b3
9  tags:
10   - ot.modbus
11   - attack.unauthorized-command
12 logsource:
13   category: network
14   product: ot
15   service: modbus
16 detection:
17   selection_write:
18     operation_type: write
19   selection_source:
20     src_ip|not_in:
21       - 192.168.1.10
22       - 192.168.1.11
23   condition: selection_write and selection_source
24 level: high
25 falsepositives:
26   - Temporary engineering activity from an unregistered maintena
```

```
Scripts > sigma-cps-modbus > data > normalized > {} sample_
1  [
2    {
3      "timestamp": "2026-01-10T10:00:00Z",
4      "src_ip": "192.168.1.99",
5      "dest_ip": "192.168.1.20",
6      "function_code": 6,
7      "operation_type": "write",
8      "unit_id": 1,
9      "address_start": 40001,
10     "address_end": 40001,
11     "quantity": 1,
12     "response_status": "success"
13   },
14   {
15     "timestamp": "2026-01-10T10:00:01Z",
16     "src_ip": "192.168.1.10",
17     "dest_ip": "192.168.1.20",
18     "function_code": 6,
19     "operation_type": "write",
20     "unit_id": 1,
21     "address_start": 40001,
22     "address_end": 40001,
23     "quantity": 1,
24     "response_status": "success"
25   },
26   {
27     "timestamp": "2026-01-10T10:00:02Z",
28     "src_ip": "192.168.1.200",
29     "dest_ip": "192.168.1.20",
30     "function_code": 16,
31     "operation_type": "write",
32     "unit_id": 1,
33     "address_start": 40005,
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   POLYGLOT NOTEBOOK

DOMENIS+adetunji@DESKTOP-3VGNO9V MINGW64 ~/Documents/Thesis/Scripts/sigma-cps-modbus
$ python -m src.main
Rule loaded, validated, translated, and executed successfully.
Title: Unauthorized Modbus Write from Non-Approved Source
Total matches: 2
[1] src_ip=192.168.1.99 operation_type=write reason=single-event match
[2] src_ip=192.168.1.200 operation_type=write reason=single-event match
```

Figure 8. Prototype execution of the unauthorized Modbus write rule, showing rule definition, normalized input events, and resulting matches.

Figure 9 shows the execution of the sensitive register range rule. The figure includes the rule definition, the normalized event data, and the resulting match produced by the prototype.

Scripts > sigma-cps-modbus > rules > sensitive_range_write.yml

```yaml
title: Modbus Write to Sensitive Register Range
id: 1a4b2f7e-0002-4c9b-a111-000000000002
status: experimental
description: Detects write operations targeting a protected register rang
author: Adeyimika Adetunji
date: 2026-01-10
references:
  - Modbus Application Protocol Specification V1.1b3
tags:
  - ot.modbus
  - attack.impact
logsource:
  category: network
  product: ot
  service: modbus
detection:
  selection_write:
    operation_type: write
    object_type: holding_register
  selection_range:
    address_start|gte: 40001
    address_end|lte: 40010
  condition: selection_write and selection_range
level: high
falsepositives:
  - Authorized maintenance activity on protected registers
```

Scripts > sigma-cps-modbus > data > normalized > sensitive_range_events.json


```json
[
  {
    "timestamp": "2026-01-10T10:05:00Z",
    "src_ip": "192.168.1.50",
    "dest_ip": "192.168.1.20",
    "function_code": 16,
    "unit_id": 1,
    "address_start": 40002,
    "quantity": 2,
    "response_status": "success"
  },
  {
    "timestamp": "2026-01-10T10:05:05Z",
    "src_ip": "192.168.1.51",
    "dest_ip": "192.168.1.20",
    "function_code": 6,
    "unit_id": 1,
    "address_start": 40020,
    "quantity": 1,
    "response_status": "success"
  },
  {
    "timestamp": "2026-01-10T10:05:10Z",
    "src_ip": "192.168.1.52",
    "dest_ip": "192.168.1.20",
    "function_code": 3,
    "unit_id": 1,
    "address_start": 40003,
    "quantity": 1,
    "response_status": "success"
  }
]
```


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POLYGLOT NOTEBOOK

```
DOMENIS+adetunji@DESKTOP-3VGNO9V MINGW64 ~/Documents/Thesis/Scripts/sigma-cps-modbus
$ python -m src.main
Rule loaded, validated, translated, normalized, and executed successfully.
Title: Modbus Write to Sensitive Register Range
Rule file: rules/sensitive_range_write.yml
Data file: data/normalized/sensitive_range_events.json
Total matches: 1
[1] src_ip=192.168.1.50 function_code=16 address_start=40002 address_end=40003 reason=single-event match
```

Figure 9. Prototype execution of the sensitive register range rule, showing rule definition, normalized event data, and resulting match.

Figure 10 presents the execution of the write exception rule. The figure shows the Sigma CPS rule definition, the normalized Modbus event set, the resulting match, and the successful automated test run supporting the correctness of the prototype pipeline.

```
Scripts > sigma-cps-modbus > rules > ! write_exception.yml
1  title: Modbus Write Request Resulting in Exception
2  id: 1a4b2f7e-0004-4c9b-a111-000000000004
3  status: experimental
4  description: Detects Modbus write operations that result in an exception
5  author: Adeyimika Adetunji
6  date: 2026-01-10
7  references:
8    - Modbus Application Protocol Specification V1.1b3
9  tags:
10   - ot.modbus
11   - attack.discovery
12 logsource:
13   category: network
14   product: ot
15   service: modbus
16 detection:
17   selection_write:
18     operation_type: write
19   selection_response:
20     response_status: exception
21   condition: selection_write and selection_response
22 level: medium
23 falsepositives:
24   - Normal configuration mismatch between client and device
```

```
Scripts > sigma-cps-modbus > data > normalized > {} write_exception_events.json > ...
1  [
2    {
3      "timestamp": "2026-01-10T10:10:00Z",
4      "src_ip": "192.168.1.60",
5      "dest_ip": "192.168.1.20",
6      "function_code": 6,
7      "unit_id": 1,
8      "address_start": 40001,
9      "quantity": 1,
10     "response_status": "exception",
11     "exception_code": 2
12   },
13   {
14     "timestamp": "2026-01-10T10:10:05Z",
15     "src_ip": "192.168.1.61",
16     "dest_ip": "192.168.1.20",
17     "function_code": 16,
18     "unit_id": 1,
19     "address_start": 40005,
20     "quantity": 2,
21     "response_status": "success"
22   },
23   {
24     "timestamp": "2026-01-10T10:10:10Z",
25     "src_ip": "192.168.1.62",
26     "dest_ip": "192.168.1.20",
27     "function_code": 3,
28     "unit_id": 1,
29     "address_start": 40003,
30     "quantity": 1,
31     "response_status": "exception",
32     "exception_code": 1
33   }
34 ]
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   POLYGLOT NOTEBOOK

DOMENIS+adetunji@DESKTOP-3VGNO9V MINGW64 ~/Documents/Thesis/Scripts/sigma-cps-modbus
$ python -m src.main
pytest -q
Rule loaded, validated, translated, normalized, and executed successfully.
Title: Modbus Write Request Resulting in Exception
Rule file: rules/write_exception.yml
Data file: data/normalized/write_exception_events.json
Total matches: 1
[1] src_ip=192.168.1.60 function_code=6 response_status=exception exception_code=2 reason=single-event match
.............                                         [100%]
13 passed in 0.09s
```

Figure 10. Prototype execution of the write exception rule, showing rule definition, normalized event set, resulting match, and automated test result.

Figure 11 shows the execution of the aggregation based rule for excessive Modbus write activity from a single source. The figure demonstrates that the prototype correctly groups matching write events and generates a detection when the configured threshold is reached within the one minute time window.

```yaml
title: Excessive Modbus Write Activity from Single Source
id: 1a4b2f7e-0005-4c9b-a111-000000000005
status: experimental
description: Detects repeated Modbus write activity from one source
author: Adeyimika Adetunji
date: 2026-01-10
references:
  - Modbus Application Protocol Specification V1.1b3
tags:
  - ot.modbus
  - attack.impact
logsource:
  category: network
  product: ot
  service: modbus
detection:
  selection_write:
    operation_type: write
  timeframe: 1m
  aggregation:
    group_by:
      - src_ip
      - dest_ip
      - unit_id
    count: 3
  condition: selection_write
level: medium
falsepositives:
  - Automated but legitimate burst-write maintenance tasks
```


```json
[
  {
    "timestamp": "2026-01-10T10:20:00Z",
    "src_ip": "192.168.1.50",
    "dest_ip": "192.168.1.20",
    "function_code": 6,
    "unit_id": 1,
    "address_start": 40001,
    "quantity": 1
  },
  {
    "timestamp": "2026-01-10T10:20:10Z",
    "src_ip": "192.168.1.50",
    "dest_ip": "192.168.1.20",
    "function_code": 6,
    "unit_id": 1,
    "address_start": 40002,
    "quantity": 1
  },
  {
    "timestamp": "2026-01-10T10:20:20Z",
    "src_ip": "192.168.1.50",
    "dest_ip": "192.168.1.20",
    "function_code": 16,
    "unit_id": 1,
    "address_start": 40003,
    "quantity": 2
  },
  {
    "timestamp": "2026-01-10T10:20:30Z",
    "src_ip": "192.168.1.99",
    "dest_ip": "192.168.1.20",
    "function_code": 6,
    "unit_id": 1,
    "address_start": 40004,
    "quantity": 1
  }
```


```
DOMENIS+adetunji@DESKTOP-3VGNO9V MINGW64 ~/Documents/Thesis/Scripts/sigma-cps-modbus
$ python -m src.main
pytest -q
Rule loaded, validated, translated, normalized, and executed successfully.
Title: Excessive Modbus Write Activity from Single Source
Rule file: rules/excessive_write.yml
Data file: data/normalized/excessive_write_events.json
Total matches: 1
[1] src_ip=192.168.1.50 dest_ip=192.168.1.20 unit_id=1 timestamp=2026-01-10T10:20:20Z reason=aggregation threshold met: 3 events within 1m for group ('192.168.1.50', '192.168.1.20', 1)
.............
13 passed in 0.14s
```

Figure 11. Prototype execution of the aggregation based excessive write rule, demonstrating threshold based repeated event detection within a one minute time window.

The unauthorized source rule produced two matches. These matches corresponded to write events whose source IP addresses were outside the configured allowlist. The allowlisted write event was correctly excluded.

The sensitive register range rule produced one match. The matched event was a write operation whose address range fell inside the protected interval. Events outside the range and events using a read function instead of a write were correctly rejected.

The write exception rule also produced one match. The matched event combined a write operation with an exception response. A successful write event and a read event with an exception response were both correctly excluded.

The aggregation rule produced one match. The detection occurred at the third write event in the relevant event group, which was the point at which the configured threshold of three matching events within one minute was reached. The observed output explicitly reported

that the aggregation threshold had been met for the specific source, destination, and unit identifier group.

In addition to rule execution, the automated test suite completed successfully. At the time of evaluation, all implemented tests passed, which confirmed expected behaviour across the core prototype components. Table 9 summarizes this result.

Table 9. Automated test summary.

Aspect	Result
Rule loading tests	Passed
Validator tests	Passed
Translator tests	Passed
Executor tests	Passed
Total implemented tests	13 passed

6.4 Analysis of findings

The evaluation results show that the prototype successfully demonstrates the core claim of the thesis: Modbus detection logic can be written in a Sigma compatible rule format, translated into executable logic, and applied to normalized telemetry in a consistent way. The rules remained readable at the authoring level, but they were still specific enough to capture practically meaningful Modbus behaviours.

One important finding is that the normalized event model was sufficient for the selected scenarios. Fields such as `operation_type`, `src_ip`, `address_start`, `address_end`, and `response_status` were enough to support all four implemented rule categories. This supports the earlier assumption that CPS telemetry can be normalized into stable fields that make portable rule authoring possible.

A second finding is that the translator design was workable in practice. The rule pipeline did not require separate native rule authoring for each detection scenario. Instead, the same structural process was applied in every case: load the rule, validate the schema, translate the selections and condition, normalize the events, and execute the resulting logic. This regularity is exactly what the thesis set out to achieve.

A third finding is that limited repeated event detection can be supported without introducing a complex correlation language. The aggregation example showed that a simple combination of `timeframe`, `group_by`, and `count` can already express a useful Modbus behaviour pattern. Although this mechanism is intentionally narrow, it strengthens the overall design because it proves that the schema is not limited to isolated single event detections.

At the same time, the evaluation also highlights the current limits of the prototype. The dataset size was small, the scenarios were controlled, and the execution target was a Python backend rather than a SIEM or industrial monitoring platform. As a result, the findings demonstrate feasibility rather than operational completeness. This limitation is acceptable within the scope of the thesis because the study is positioned as a proof of concept rather than as a finished deployment framework.

Overall, the evaluation supports the thesis hypothesis. The proposed Sigma CPS Modbus approach improved the structure and logical portability of the implemented detections while remaining understandable and executable. Within the defined scope, the results provide evidence that a Sigma compatible rule abstraction for normalized Modbus telemetry is technically feasible and practically meaningful as a proof of concept.

7 Discussion

This chapter interprets the evaluation results, discusses the practical value of the proposed Sigma CPS Modbus approach, and defines the main limitations of the work. The purpose of the discussion is not only to restate the results, but to explain what was actually demonstrated, what remains outside the scope of the current prototype, and how the work can be extended in the future.

7.1 Interpretation of results

The evaluation results show that the proposed Sigma CPS Modbus approach is feasible within the scope of this thesis. The prototype successfully loaded Sigma CPS Modbus rules, validated them against the defined schema, translated them into executable Python based logic, and applied them to normalized Modbus telemetry. This means that the central claim of the thesis was supported in practice: Modbus detection logic can be expressed in a Sigma compatible format and processed systematically instead of being written only as platform specific native rules.

The evaluation also shows that the proposed design is not limited to one narrow type of rule. The prototype handled source based detections, address based detections, response based detections, and a simple repeated behaviour detection using aggregation. This is important because it shows that the rule model is expressive enough to support more than trivial exact match cases. At the same time, the supported rule types remained understandable and implementation friendly, which supports the original design goal of preserving readability.

A further interpretation of the results is that the Sigma CPS Modbus model works well as an intermediate layer between telemetry and execution logic. The rules were written in a structured, portable form, while the actual execution was handled separately by the translator and execution engine. This separation reflects the main architectural value of the approach. The rule author does not need to write detection logic directly in native backend syntax. Instead, the backend specific execution can be derived from a single higher level

rule description.

7.2 Readability and logical portability of the approach

One of the main motivations behind the thesis was that Modbus detection logic is often embedded in scripts, engine specific rules, or implementation specific logic that is difficult to reuse across environments. The prototype suggests that a Sigma compatible representation can improve this situation by making the rule logic easier to read and easier to separate from the final execution target. In the implemented examples, fields such as `operation_-type`, `src_ip`, `unit_id`, `address_start`, `address_end`, and `response_status` made the rules understandable without requiring the reader to inspect low level packet data or backend specific syntax.

The portability claim of the thesis should, however, be interpreted carefully. The prototype demonstrated *logical portability*, meaning that the same rule structure could be reused and translated in a consistent way inside the implemented system. It did not yet demonstrate full operational portability across multiple production SIEMs or industrial monitoring platforms. In a complete ecosystem, Sigma CPS rules would still need backend specific mappings so that the same high level rule could be emitted in the syntax expected by the target environment. In that sense, the current work should be understood as proving that such translation is structurally possible and that the main remaining step is target specific mapping rather than redesigning the rule itself [7, 6].

This is an important distinction. The prototype does not claim that Modbus rules can already be dropped directly into every SIEM without additional work. Instead, it shows that the detection can be authored once in a Sigma compatible form and then translated in a systematic way. That is still a meaningful result because it reduces repeated rule authoring and makes the detection logic easier to manage over time.

7.3 Practical usefulness

From a practical perspective, the prototype is useful because it shows a realistic workflow for portable Modbus detection engineering. In this workflow, telemetry is first normalized into a stable event model, a Sigma CPS rule is written over those fields, and the rule is then

translated into executable logic. This is a practical architecture because it separates three concerns that are often mixed together in industrial monitoring: parsing, rule definition, and execution.

The implemented rule examples also show that the approach is suitable for detections that security teams would reasonably want to express. Unauthorized write operations, writes to protected address ranges, write requests resulting in exception responses, and repeated write behaviour are all realistic Modbus monitoring cases. The fact that these could be expressed and executed through a single schema supports the practical value of the proposed model.

The usefulness of the system also lies in what it avoids. Without a portable rule layer, the same Modbus detection would often need to be rewritten separately for each monitoring environment. Even in this simplified proof of concept implementation, the benefit of abstraction is already visible. The detection logic is stored once in YAML, while the system handles validation, translation, and execution in separate stages.

7.4 Limitations and threats to validity

The thesis has several important limitations.

First, the prototype targets Python based executable logic rather than a production SIEM backend. This was a deliberate design decision. The goal of the implementation was to prove that Sigma CPS Modbus rules can be translated into executable logic in a systematic way. Full integration with multiple operational backends would have substantially increased scope and would have shifted the thesis away from the core proof of concept objective. Therefore, the prototype should be interpreted as a backend neutral architectural demonstration with one concrete execution target.

Second, the thesis assumes the availability of normalized Modbus telemetry and does not treat raw packet parsing or telemetry ingestion as the main research contribution. This was also deliberate. In practical monitoring environments, parsing and ingestion are typically handled by existing collectors, protocol parsers, or preprocessing pipelines. Re implementing a full normalization pipeline would have duplicated that functionality and moved the thesis away from its actual focus, which is the rule representation and translation

layer. For that reason, normalization was treated as an architectural prerequisite rather than as the main artifact under study. This assumption is consistent with parser first industrial monitoring approaches and with tooling that exposes parsed Modbus telemetry in more structured form for downstream analysis [15, 16, 14].

Third, the evaluation was based on controlled normalized event sets and automated tests, not on a full public dataset benchmark. This means the evaluation demonstrates correctness and feasibility of the prototype pipeline, but not broad operational performance across diverse real world traffic. The prototype shows that the designed rule model works on representative scenarios, but additional work would be needed to evaluate it on larger public datasets or directly on parser generated telemetry [19].

Fourth, the aggregation support is intentionally narrow. The prototype can express simple repeated event behaviour using a timeframe, grouping fields, and a count threshold, but it does not implement a full temporal correlation language. This limits the complexity of behaviours that can currently be described.

There are also threats to validity. The main construct validity risk is that the chosen scenarios may not cover the full diversity of Modbus detection needs. The main internal validity risk is that the controlled event sets were designed for proof of concept testing and therefore may not reflect all edge cases of operational traffic. The main external validity risk is that the prototype was tested in a simplified Python execution environment rather than in multiple operational platforms. These threats do not invalidate the thesis, but they do define the boundary of the conclusions that can be drawn.

7.5 Future directions

Several future extensions follow naturally from this work.

The first is the implementation of additional backends. Since the prototype already proves that Sigma CPS Modbus rules can be parsed and translated into executable logic, the next step would be to add concrete target mappers for SIEM platforms, Zeek based monitoring outputs, or other rule execution environments [7, 14, 11].

The second is tighter integration with protocol parsing and telemetry ingestion pipelines.

Although normalization was intentionally treated as out of scope in this thesis, future work could connect the Sigma CPS rule model directly to parser generated Modbus telemetry from existing monitoring tools [15, 16, 14].

The third is broader evaluation on public datasets and larger traffic collections. Public datasets such as CIC Modbus 2023 would support more varied traffic evaluation [19]. In addition, testbeds and emulation environments such as SWaT and MiniCPS could support broader experimental validation of the approach in more realistic ICS settings [27, 28].

The fourth is extending the schema beyond Modbus to additional CPS protocols. The current thesis narrowed the implementation to Modbus for methodological reasons, but the same architectural idea could later be explored for other protocol families. This still remains aligned with the broader CPS motivation defined earlier in the thesis.

Overall, the discussion shows that the work makes a focused but meaningful contribution. It does not provide a complete operational detection framework, but it does demonstrate that a Sigma compatible, protocol aware rule layer for Modbus is both technically feasible and practically useful as a proof of concept.

8 Summary

This thesis addressed the problem of how to extend Sigma style detection engineering to Modbus based cyber physical system telemetry. The central challenge identified in the work was that CPS detections are often implemented in tool specific forms, which makes them harder to reuse, translate, and maintain across monitoring environments. To address this problem, the thesis proposed a Sigma compatible rule model for normalized Modbus telemetry and implemented a prototype that parses, validates, translates, and executes such rules over structured events.

The work was deliberately narrowed to Modbus in order to keep the study technically feasible while still testing the core research idea. This narrowing did not change the overall topic of extending Sigma detection rules toward CPS telemetry. Instead, it provided a focused and defensible way to design and evaluate the rule model in one representative protocol domain.

The main outcome of the thesis is a proof of concept showing that Modbus detection logic can be authored once in a Sigma compatible form and then translated into executable logic through a structured workflow. The implemented prototype included a normalized Modbus event model, a Sigma CPS Modbus rule schema, a rule loader, a validator, a translator, a normalizer, and an execution engine. The evaluation showed that the prototype correctly handled representative rule types such as unauthorized writes, writes to protected register ranges, write requests resulting in exceptions, and repeated write activity from a single source. These results support the argument that portable Modbus detection engineering is technically feasible within a Sigma compatible design.

The first research question asked how the Sigma rule schema can be extended to represent Modbus telemetry while preserving readability and simplicity. The thesis answered this by defining a Sigma compatible rule structure that keeps familiar Sigma elements such as metadata, `logsource`, selections, and conditions, while extending the detection vocabulary

with Modbus aware normalized fields such as `function_code`, `operation_type`, `unit_-id`, `address_start`, `address_end`, and `response_status`. This showed that Modbus specific semantics can be integrated without abandoning the overall readability of the Sigma model.

The second research question asked how Sigma CPS Modbus rules can be translated into executable detection logic. The thesis answered this by implementing a prototype translation workflow in which YAML rules are loaded, validated, converted into an internal representation, translated into executable Python based predicates, and executed over normalized Modbus events. The implementation demonstrated that translation can be systematic and does not require each detection to be rewritten directly in native backend syntax.

The third research question asked how effective and practically useful the proposed approach is for representing and detecting representative Modbus misuse scenarios. Within the scope of the prototype, the answer is positive. The implemented rules executed correctly on the prepared normalized event sets, and the prototype supported both single event detections and limited repeated event detections through aggregation. At the same time, the thesis also showed that the current result should be interpreted as a feasibility demonstration rather than as a full operational benchmark across production platforms.

The thesis does not claim to solve CPS detection engineering in general, nor does it claim that Modbus rules can already be deployed directly into every target system without additional mapping. Instead, its contribution is narrower and more practical: it demonstrates that there can be a portable rule layer between normalized Modbus telemetry and backend specific execution logic. This is an important result because it suggests that much of the difficulty in cross platform Modbus detection engineering lies not in the detection idea itself, but in the lack of a reusable intermediate representation.

Several directions for future work follow naturally from this result. One is the implementation of concrete output backends for SIEMs or other monitoring platforms. Another is integration with parser generated Modbus telemetry from existing ingestion pipelines. A third direction is broader validation using larger public datasets or operational traffic collections. Finally, the same architectural approach could be explored for other CPS

protocols after the Modbus proof of concept established in this thesis.

Overall, the thesis supports the hypothesis that a Sigma compatible extension for Modbus telemetry can improve the structure, readability, and logical portability of CPS detection logic while remaining implementable in a working prototype. Within the defined scope, the work provides a concrete step toward more portable detection engineering for cyber physical systems.

References

[1] Keith Stouffer et al. *Guide to Operational Technology (OT) Security*. NIST Special Publication 800-82 Revision 3. National Institute of Standards and Technology, 2023. DOI: 10.6028/NIST.SP.800-82r3. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r3.pdf (visited on 04/21/2026).

[2] SigmaHQ. *Sigma Rules Specification*. 2025. URL: https://github.com/SigmaHQ/sigma-specification/blob/main/specification/sigma-rules-specification.md (visited on 04/21/2026).

[3] SigmaHQ Community. *pySigma Documentation*. 2026. URL: https://sigmahq-pysigma.readthedocs.io/ (visited on 04/21/2026).

[4] Robert Mitchell and Ing-Ray Chen. „A Survey of Intrusion Detection Techniques for Cyber-Physical Systems“. In: *ACM Computing Surveys* 46.4 (2014), 55:1–55:29. DOI: 10.1145/2542049.

[5] Yan Hu et al. „A survey of intrusion detection on industrial control systems“. In: *International Journal of Distributed Sensor Networks* 14.8 (2018). DOI: 10.1177/1550147718794615.

[6] SigmaHQ Community. *pySigma Documentation: Processing Pipelines*. 2026. URL: https://sigmahq-pysigma.readthedocs.io/en/latest/Processing_Pipelines.html (visited on 04/21/2026).

[7] SigmaHQ Community. *pySigma Documentation: Backends*. 2026. URL: https://sigmahq-pysigma.readthedocs.io/en/latest/Backends.html (visited on 04/21/2026).

[8] MODBUS Organization. *MODBUS Application Protocol Specification V1.1b3*. Apr. 26, 2012. URL: https://www.modbus.org/file/secure/modbusprotocolspecification.pdf (visited on 04/21/2026).

[9] Niv Goldenberg and Avishai Wool. „Accurate Modeling of Modbus/TCP for Intrusion Detection in SCADA Systems“. In: *International Journal of Critical Infrastructure Protection* 6.2 (2013), pp. 63–75. DOI: 10.1016/j.ijcip.2013.05.001.

[10] Thomas H. Morris et al. „Deterministic Intrusion Detection Rules for MODBUS Protocols“. In: *Proceedings of the 46th Hawaii International Conference on System Sciences*. 2013, pp. 1773–1781. DOI: 10.1109/HICSS.2013.174.

[11] Open Information Security Foundation. *Suricata Documentation: Rules Format*. 2026. URL: https://docs.suricata.io/en/latest/rules/intro.html (visited on 04/21/2026).

[12] Digital Bond. *Quickdraw-Snort: ICS IDS/IPS Rules for Snort*. 2026. URL: https://github.com/digitalbond/Quickdraw-Snort (visited on 04/22/2026).

[13] Digital Bond. *Quickdraw-Suricata: ICS IDS Rules for Suricata*. 2026. URL: https://github.com/digitalbond/Quickdraw-Suricata (visited on 04/22/2026).

[14] Zeek Project. *Zeek Modbus Analyzer and Logging Interface*. 2026. URL: https://docs.zeek.org/en/lts/scripts/base/protocols/modbus/main.zeek.html (visited on 04/22/2026).

[15] Cybersecurity and Infrastructure Security Agency. *ICSNPP: Industrial Control Systems Network Protocol Parsers*. 2026. URL: https://github.com/cisagov/ICSNPP (visited on 04/21/2026).

[16] Cybersecurity and Infrastructure Security Agency. *ICSNPP-Modbus*. 2026. URL: https://github.com/cisagov/ICSNPP-Modbus (visited on 04/21/2026).

[17] Jakub Suchorab, Sebastian Plamowski, and Maciej Ławryńczuk. „Anomaly Detection System for Modbus Data Based on an Open Source Tool". In: *Computers & Security* 157 (2025), p. 104572. DOI: 10.1016/j.cose.2025.104572.

[18] MODBUS Organization. *Modbus Specifications and Implementation Guides*. 2026. URL: https://www.modbus.org/modbus-specifications (visited on 04/21/2026).

[19] Canadian Institute for Cybersecurity. *CIC Modbus Dataset 2023*. 2023. URL: https://www.unb.ca/cic/datasets/modbus-2023.html (visited on 04/21/2026).

[20] SigmaHQ. *Sigma: Main Sigma Rule Repository*. 2025. URL: https://github.com/SigmaHQ/sigma (visited on 04/21/2026).

[21] SigmaHQ. *Sigma Correlation Rules Specification*. 2025. URL: https://github.com/SigmaHQ/sigma-specification/blob/main/specification/sigma-correlation-rules-specification.md (visited on 04/21/2026).

[22] Tirthankar Ghosh et al. „Anomaly Detection for Modbus over TCP in Control Systems Using Entropy and Classification-Based Analysis". In: *Journal of Cybersecurity and Privacy* 3 (2023), pp. 895–913. DOI: 10.3390/jcp3040041. URL: https://www.mdpi.com/journal/jcp (visited on 04/21/2026).

[23] Zeek Project. *Zeek Documentation: Notice Framework*. 2026. URL: https://docs.zeek.org/en/master/frameworks/notice.html (visited on 04/21/2026).

[24] Ken Peffers et al. „A Design Science Research Methodology for Information Systems Research". In: *Journal of Management Information Systems* 24.3 (2007), pp. 45–77. DOI: 10.2753/MIS0742-1222240302.

[25] Adeyimika Agboola Adetunji. *MSC-Thesis: Sigma-CPS Modbus Prototype*. GitHub repository. Accessed: 2026-04-21. 2026. URL: https://github.com/Weirdlogic/MSC-Thesis.

[26] Tiago Cruz and collaborators. *ICS_PCAPS Release: MODBUS/TCP#1*. GitHub release page describing the MODBUS/TCP capture set and testbed. 2025. URL: https://github.com/tjcruz-dei/ICS_PCAPS/releases/tag/MODBUSTCP%5C%231 (visited on 04/21/2026).

[27] Aditya P. Mathur and Nils Ole Tippenhauer. „SWaT: A Water Treatment Testbed for Research and Training on ICS Security“. In: *Proceedings of the 2016 International Workshop on Cyber-physical Systems for Smart Water Networks*. 2016, pp. 31–36. DOI: 10.1109/CySWater.2016.7469060.

[28] Daniele Antonioli and Nils Ole Tippenhauer. „MiniCPS: A Toolkit for Security Research on CPS Networks“. In: *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or Privacy*. 2015, pp. 91–100. DOI: 10.1145/2808705.2808715.

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Adeyimika Agboola Adetunji

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Extending Sigma Detection Rules for Cyber-Physical Systems", supervised by Muaan Ur Rehman
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright
2. I am aware that the author also retains the rights specified in clause 1 of the nonexclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive licence shall not be valid for the period.

Appendix 2 – Function-code mapping and normalization details

This appendix presents the function-code mapping used by the Sigma-CPS Modbus normalizer and summarizes the exception-response semantics relevant to the prototype. Function names are taken from the official MODBUS Application Protocol Specification V1.1b3, while the higher-level fields `operation_type`, `object_type`, and `is_write` are author-defined normalization fields derived from those protocol semantics [8].

The mapping in Table 10 is the one used by the prototype normalizer to derive semantic fields from raw Modbus function codes. The current implementation focuses on Modbus/TCP use cases relevant to the prototype rules described in the main text.

Table 10. Function-code mapping used by the Sigma-CPS Modbus normalizer.

Function code	Function name	operation_-type	object_type	is_write
1	Read Coils	read	coil	false
2	Read Discrete Inputs	read	discrete_input	false
3	Read Holding Registers	read	holding_register	false
4	Read Input Registers	read	input_register	false
5	Write Single Coil	write	coil	true
6	Write Single Register	write	holding_register	true
15	Write Multiple Coils	write	coil	true
16	Write Multiple Registers	write	holding_register	true
22	Mask Write Register	write	holding_register	true
23	Read/Write Multiple Registers	read_write	holding_register	true
24	Read FIFO Queue	read	fifo_queue	false
43	Encapsulated Interface Transport	encapsulated	other	false

The current prototype does not rely on every public function code defined by the specification. In particular, the implementation concentrates on the read and write functions most

relevant to the selected Modbus misuse scenarios. Some functions documented in the specification, such as `Read Exception Status`, `Diagnostics`, `Get Comm Event Counter`, `Get Comm Event Log`, and `Report Server ID`, are marked by the standard as Serial Line only and are therefore not central to the present Modbus/TCP-focused prototype [8].

The prototype also relies on normalized response status rather than raw exception function codes in the rule layer. In Modbus, an exception response is represented by returning the original function code with the most significant bit set, together with an exception code that explains the error condition [8]. Table 11 lists the most relevant exception codes for the current implementation.

Table 11. Common Modbus exception codes relevant to the prototype.

Code	Name	Interpretation in the prototype context
1	Illegal Function	The target device does not support the requested function.
2	Illegal Data Address	The request references an invalid or unauthorized address.
3	Illegal Data Value	The request value is syntactically valid but not acceptable for the operation.
4	Server Device Failure	The device encountered an internal failure while processing the request.

Within the prototype, these protocol semantics are abstracted into normalized fields such as `response_status` and `exception_code`. This design choice keeps the rule language readable while preserving the security-relevant meaning of the underlying Modbus protocol

behaviour.

Appendix 3 – Prototype artifacts and selected outputs

This appendix provides representative artifacts from the Sigma-CPS Modbus prototype. The purpose of this appendix is to support reproducibility and to show concrete examples of the rule files, normalized event data, and selected execution outputs discussed in Chapters 5 and 6.

The complete prototype source code is provided in the accompanying GitHub repository [25]. The repository contains the Python implementation modules, Sigma CPS Modbus rule files, normalized event samples, selected execution outputs, and automated tests used during the thesis evaluation. The code is provided as a proof-of-concept implementation supporting the design and evaluation presented in Chapters 5 and 6.

Representative Sigma-CPS Modbus rule

Listing 1 shows the full YAML rule used to detect write-capable Modbus requests from non-approved source hosts.

Listing 1. Full Sigma-CPS Modbus rule for unauthorized write requests.

```
title: Unauthorized Modbus Write from Non-Approved Source
id: 1a4b2f7e-0001-4c9b-a111-000000000001
status: experimental
description: Detects Modbus write requests from sources outside the
    approved host list.
author: Adeyimika Adetunji
date: 2026-01-10
references:
  - MODBUS Application Protocol Specification V1.1b3
tags:
  - ot.modbus
  - attack.unauthorized-command
logsource:
  category: network
  product: ot
```

```
  service: modbus
detection:
  selection_write:
    operation_type: write
  selection_source:
    src_ip|not_in:
      - 192.168.1.10
      - 192.168.1.11
  condition: selection_write and selection_source
level: high
falsepositives:
  - Temporary engineering activity from an unregistered maintenance
      host
```

Representative normalized event data

Listing 2 shows a representative normalized event set used to test the aggregation rule for repeated write activity.

Listing 2. Representative normalized Modbus events for aggregation testing.


```
[
  {
    "timestamp": "2026-01-10T10:20:00Z",
    "src_ip": "192.168.1.50",
    "dest_ip": "192.168.1.20",
    "function_code": 6,
    "unit_id": 1,
    "address_start": 40001,
    "quantity": 1
  },
  {
    "timestamp": "2026-01-10T10:20:10Z",
    "src_ip": "192.168.1.50",
    "dest_ip": "192.168.1.20",
    "function_code": 6,
    "unit_id": 1,
    "address_start": 40002,
    "quantity": 1
```

```
  },
  {
    "timestamp": "2026-01-10T10:20:20Z",
    "src_ip": "192.168.1.50",
    "dest_ip": "192.168.1.20",
    "function_code": 16,
    "unit_id": 1,
    "address_start": 40003,
    "quantity": 2
  },
  {
    "timestamp": "2026-01-10T10:20:30Z",
    "src_ip": "192.168.1.99",
    "dest_ip": "192.168.1.20",
    "function_code": 6,
    "unit_id": 1,
    "address_start": 40004,
    "quantity": 1
  }
]
```


Representative execution output

Listing 3 shows a representative execution output from the aggregation-based rule. It illustrates how the prototype reports the threshold-based match once the required number of write events has been observed within the configured time window.

Listing 3. Representative execution output for the aggregation-based repeated-write rule.

```
Rule loaded, validated, translated, normalized, and executed
   successfully.
Title: Excessive Modbus Write Activity from Single Source
Rule file: rules/excessive_write.yml
Data file: data/normalized/excessive_write_events.json
Total matches: 1
[1] src_ip=192.168.1.50 dest_ip=192.168.1.20 unit_id=1 timestamp
   =2026-01-10T10:20:20Z reason=aggregation threshold met: 3 events
    within 1m for group ('192.168.1.50', '192.168.1.20', 1)
.............
```

```
13 passed in 0.09s
```

Representative protocol evidence

Figures in the main text already show author-created Wireshark screenshots based on the ICS_PCAPS MODBUS/TCP#1 packet captures [26]. Those figures complement the normalized event examples presented here by showing the packet-level protocol evidence from which the normalization layer derives structured fields such as transaction identifier, function code, address information, and register values.