JOURNAL OF MECHANICS OF CONTINUA AND
MATHEMATICAL SCIENCES
www.journalimcms.org

ISSN (Online) : 2454 -7190 Vol.-21, No.-02, February (2026) pp 41-58 ISSN (Print) 0973-8975

REAL-TIME DETECTION OF MALICIOUS LOGIC INJECTION IN SCADA SYSTEMS USING HYBRID YARA SIGNATURES

Gulab Kumar Mondal[1], Arijit Das[2], Moumita Pal[3], Biswarup Neogi[4] DharamPal Singh[5]

[1]JIS University, ISOAH Data Securities Pvt LtdWest Bengal, India.

[2] Computer Science & Technology, JIS College of Engineering, West Bengal, India.

[3]Head of the Department, Electronics and Communication Engineering, JIS College of Engineering, West Bengal, India.

[4]JIS University; Hazi A.K. Khan College, West Bengal, India.

[5]Computer Science and Engineering, JIS University, West Bengal, India.

Email: [1]gulabmondal121@gmail.com, [2]webstararijit@gmail.com, [3]moumitajisceece@gmail.com, [4]biswarupneogi@gmail.com, [5]dharmpal1982@gmail.com

Corresponding Author: **Biswarup Neogi**

https://doi.org/10.26782/jmcms.2026.02.00003

(Received: December 03, 2025; Revised: January 26, 2026; Accepted : February 07, 2026)

Abstract:

Modern Industrial Control Systems (ICS) and Supervisory Control and Data Acquisition (SCADA) networks face a growing class of logic-layer attacks in which adversaries silently manipulate configuration or project files instead of deploying traditional malware. Existing defences, such as network intrusion detection systems and machine-learning-based anomaly detectors, struggle to observe these pre-deployment logic changes and often incur high operational complexity. This paper presents a lightweight, host-based framework that uses YARA, a rule-based pattern-matching engine, to perform static inspection of XML configuration files generated by SCADA engineering tools. The proposed system is implemented on a Windows 10 engineering workstation using ModbusPal as a Modbus TCP simulator, Python for file monitoring and GUI development, and YARA CLI/Python bindings for rule execution. Custom YARA rules are crafted to detect unauthorized Modbus function code 5 (Write Single Coil) operations targeting critical coil addresses, modelling malicious logic injections such as covert actuator activations. In a controlled lab environment, using a variety of ModbusPal project files, a combination of benign (no infiltration) and tampered project

files, as well as our detection framework, achieved less than 200 milliseconds of latency for detecting true positives (and 0 false positives and 0 false negatives) for the defined ruleset and under a negligible resource overhead.

These findings indicate that static logic validation at the host-level would fulfil an effective integrity pre-deployment check for PLC logic in addition to current network-based and behaviour-based ICS security mechanisms, without requiring modification of the installed PLC hardware and network protocol.

Keywords: SCADA Security, Industrial Control Systems (ICS), YARA, Logic, Modbus, TCP, Host-Based Intrusion Detection, Static Analysis, OT Cybersecurity

I. Introduction

Manufacturing, energy, and other infrastructures today require Industrial Control Systems (ICS). The core of an ICS is a Supervisory Control and Data Acquisition (SCADA) system that enables people to see the processes happening in their manufacturing/utility environment, gather data on it, and remotely control various devices (pumps, valves, actuators, etc.) involved in the industrial processes. Historically, SCADA systems were built on proprietary protocols and hardware and were therefore isolated from cyberattacks due to being air-gapped (not connected to the Internet). Industry 4.0 has brought about the connection of IT and OT industries, allowing SCADA systems to become more vulnerable to cyber threats (from within the OT environment and external attackers).

Logic manipulation attacks are not like typical IT attacks, where someone seeks to gain access to or control over hardware or software by attacking either system binaries or network traffic. Instead, logic manipulation attacks alter the control layer of a system - i.e., the configuration layer (modification of the program or project files used to control a PLC and the control logic used by the PLC), and/or XML-based descriptors associated with that logic, causing them to function as desired by the attacker and not as intended by the designer. Recent threat intelligence reports (e.g., Dragos's assessment [IV] of the INCONTROLLER (PIPEDREAM) toolkit and FrostyGoop malware) provide evidence demonstrating that malicious actors can embed logic-layer payloads directly into engineering project files to evade traditional cyber defences, such as firewalls and network intrusion detection systems.

Although much of the initial work, including the rule sets published by ICS-CERT for BlackEnergy, focuses on detecting executable code, limited effort has been put forth to develop signature-based tools to verify the integrity and authenticity of static logic in configuration files for industrial control systems (ICS). Our work expands upon existing research to provide new uses for YARA beyond signatures for executables by extending its applicability to the logic layer of configuration files within the ICS world.

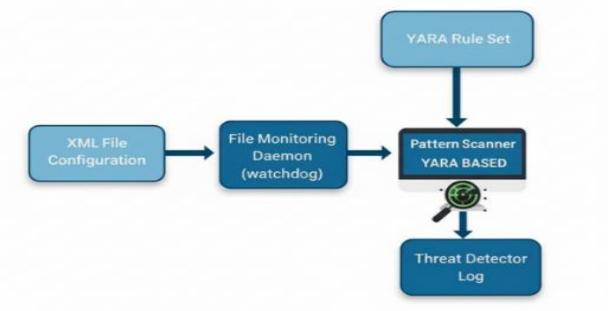


Fig. 1. Architectural Diagram

This study provides a means to use YARA as a lightweight detection mechanism to statically analyze the logic within XML files created by SCADA engineering tools and simulators, such as ModbusPal. We validate the proposed approach by constructing specific attack scenarios in a laboratory setting using ModbusPal to emulate control of PLCs, writing custom YARA rules to identify malicious payloads, and validating detection results through the use of both benign and malicious configuration files. Overall, the final project that resulted from this research has produced a system that is fast, effective, and easily incorporated into an organization's normal operating procedures; this product will significantly enhance organizations' overall cybersecurity posture in ICS/SCADA environments.

II. Literature Review

As a result of the increased threats of external cyberattacks on Industrial Control Systems (ICS) and Supervisory Control and Data Acquisition (SCADA) systems in recent years, there has been increased interest in the security of these systems. Traditional security measures rely heavily on network-layer security tools (Zeek, Snort, Suricata) for detecting anomalies in network traffic based on a signature-based method of deep packet inspection with a few known limitations. The reliance on these types of tools has resulted in significant blind spots caused by encryption, lack of visibility into host-level events, and segmentation of networks [VIII], [V]. Researchers have recently started to use machine learning (ML) techniques as a way to address these blind spots. For example, convolutional neural networks (CNNs) have been applied to detect anomalous patterns of Modbus TCP traffic [XIV], and researchers have developed the SCAPHY framework to correlate physical traces of malicious activity with significant improvements in the accuracy of detecting anomalies. However, most ML-based solutions still suffer from high false-positive rates and require frequent retraining, making them impractical for most ICS systems in continuously changing environments [XI], [XIV].

ICS malware is on the rise. Some recent examples include INCONTROLLER (or PIPEDREAM) and FrostyGoop. These types of malware do not just infect computers; they are embedded into engineering project files (or con-figuration file) and therefore will bypass traditional network security systems [VI] [VII]. In addition, a recent study performed by Forescout Labs showed how an attacker could use an unauthorized coil

write operation over Modbus to gain access to critical physical assets without alerting a traditional network intrusion detection system [X]. The integrity of the developed PLC (Programmable Logic Controller) logic should be ensured. However, although static host-level analysis may provide some level of coverage, the full capabilities that these PLC development environments (for example, Velocio's vBuilder or Rockwell's Studio 5000) can offer are still being missed. The project files that are generated from PLC development environments (i.e., vlp, l5x, .xml) contain comprehensive executable logic, and if altered by an insider or corrupted PC, these files could allow stealthy malware, like LogicLocker [XXI], to be installed.

In recent years, many organizations have developed tools that make use of YARA to identify and analyse the malware associated with known ICS malware families: For example, ICS-CERT has produced YARA rules that identify the malware associated with BlackEnergy and Havex. However, these tools were not intended for application to PLC project logic files and only provided a foundation for subsequent efforts to make use of YARA Rules for Static Logic Verification of XML-Based Configuration Files [XII], [XIII]. These new efforts have built upon the initial work done by the authors from Nguyen et al. [XX], who proposed the use of a lightweight hash tree for verifying the integrity of PLC Files. Furthermore, although both ICS-Rank and SCADAhunt provide a system-wide scoring system for ICS risk, as well as the identification of anomalies, neither framework has produced a lightweight, pre-deployment static logic validation tool [II], [I].

III. Threat Model (SCADA Logic Injection)

This paper adopts a SCADA-specific adversarial threat model to clearly define attacker capability, injection scope, and the class of malicious logic injection events targeted by the proposed hybrid YARA signatures. The objective of the adversary is to introduce unauthorized logic into engineering configuration artifacts such that critical Modbus coil activation is executed without satisfying expected authorization conditions, resulting in unsafe actuator behaviour prior to field deployment.

A. Attacker Access Vectors.

We assume the attacker can reach the SCADA project files through one or more realistic access paths: (i) compromise of the engineering workstation (e.g., malware or remote access) enabling unauthorized edits to ModbusPal XML project files, (ii) insider modification where a trusted user directly inserts malicious FC05 write-coil instructions, (iii) supply-chain tampering where the delivered project file is modified before reaching the site, and (iv) file delivery mechanisms such as USB drop or email attachment where a compromised XML file is placed into the monitored watch directory.

B. Injection Phase.

The scope of this work focuses on offline (pre-deployment) logic injection, where malicious coil-write directives are embedded into stored configuration artifacts before being uploaded to operational PLCs. Online runtime modification (e.g., altering PLC

execution during live operations) is considered out of scope for this framework because the proposed scanner performs static inspection of files at rest.

C. Targeted Logic Types.

SCADA logic injection can target multiple layers. In this work, the detection focus is on the engineering tool project XML, specifically ModbusPal XML configuration files used during simulation and pre-deployment workflows. Attacks targeting PLC program logic (ladder logic, Structured Text, Function Block Diagrams) are considered a future extension, while HMI scripts and supervisory logic are treated as out-of-scope or future scope due to differing artifact formats and execution semantics.

D. Trigger Mechanisms.

The adversarial payload may be designed to activate under different triggering conditions, including time-triggered, sensor-triggered, manual-triggered, or rare-event-triggered conditions. While the proposed method does not execute or simulate runtime state transitions, the presence of an embedded unauthorized coil-write directive to critical coil addresses is treated as a high-risk pre-deployment integrity violation independent of its intended trigger condition.

E. Mapping to Detection Coverage.

Accordingly, the hybrid YARA engine is designed to detect offline XML-based coil write injection patterns, particularly unauthorized Modbus FC05 write-single-coil operations targeting critical coil addresses (e.g., 40010/40011 in the evaluation setup). The approach does not fully detect runtime stealth logic that may be encoded inside PLC binaries or executed through live network-triggered manipulation without file modification. Therefore, the framework is positioned as a deterministic pre-deployment integrity gate for engineering configuration artifacts, complementing existing runtime monitoring approaches.

IV. Operational Methodology

A. Environment & Simulation Setup

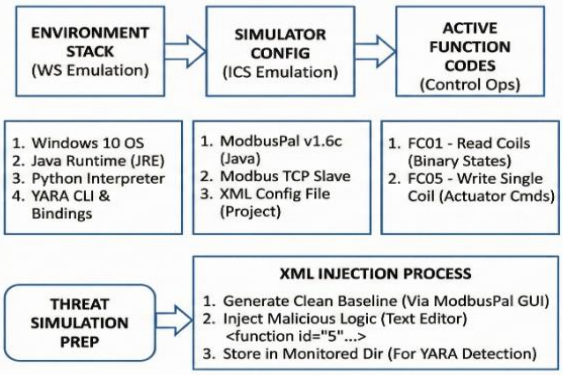


Fig. 2. Environment & Simulation Setup

The Environment and Simulation Setup consists of Modbus Coil Data Acquisition and a simulated industrial control system (ICS) environment configured to evaluate the proposed YARA-based static detection approach for Supervisory Control and Data Acquisition (SCADA) logic manipulation. The laboratory environment was designed to closely emulate a typical engineering workstation used by control engineers during pre-deployment interactions with field Programmable Logic Controllers (PLCs). To ensure setup repeatability and mitigate compatibility issues, the test environment was established on a Windows 10 operating system with a stable and widely adopted ICS software stack, including the Java Runtime Environment (JRE) and a Python interpreter configured with YARA Command Line Interface (CLI) tools and Python bindings. PLC behaviour was simulated using ModbusPal version 1.6c, a Java-based Modbus TCP slave emulator capable of performing read/write operations on key memory elements such as coils, discrete inputs, input registers, and holding registers. Coil reading and single-coil write operations can be simulated using function codes FC05 and FC01. By providing realistic control functionality (including reading binary coil states and sending commands to turn on motors/valves to the PLC), ModbusPal allows users to create XML project files and simulate attack scenarios. To do this, malicious programming instructions (i.e., logic instructions) were added to the XML project file (e.g., <function id = "5" address = "40010" value = "1"/> and <function id = "5" address = "40011" value = "1"/>). These malicious programming instructions indicate which coils to activate when a PLC program is running. To simulate a threat, a standard XML project file was created with ModbusPal's graphical user interface (GUI). A threat was simulated by inserting malicious programming instructions (to represent an insider attack or malware) using a text editor and then storing modified XML project files in a monitored project directory for comparison to the original XML project file to assess the ability of a static detection mechanism to identify pre-deployment programming changes.

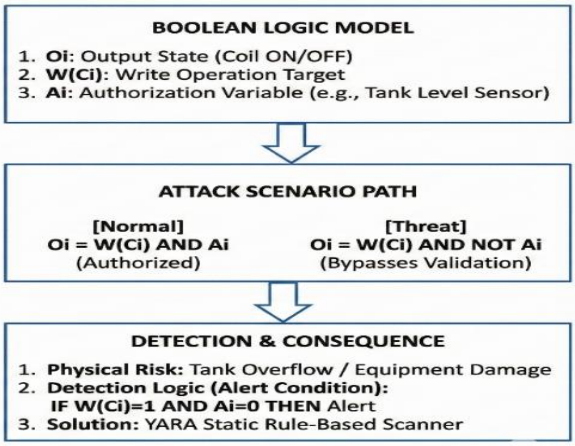


Fig. 3. Logic-Layer Threat Modelling

B. Logic-Layer Threat Modelling

Using an established threat model for Modbus-based control systems, we create a logical representation of how an unauthorized write on a Modbus coil would violate the established logical controls and energize a Modbus coil. For example, if there is an unauthorized write to coil i of $W(C_i)$, then the output state of the coil O_i, which indicates its current state (ON or OFF), will be: $O_i = W(C_i) \wedge \neg A_i$, which implies that A_i is false. This means that the coil was energized when $W(C_i)$ was received, but since A_i is false, W(Ci) was not a valid authorized write according to the logical controls prescribed. In a properly authorized configuration, an authorized write must exist before the coil can be energized; therefore, we represent an authorized write as $O_i = W(C_i) \wedge A_i$. Conversely, when a bad actor injects malicious logic that modifies the Modbus configuration (for example, by including ModbusPal-generated XMLs that include coil write requests), the bad actor can write to coil i directly without going through the controls and checks associated with the predefined logic used to validate the write. Therefore, $W(C_i) = 1$ will be equal to one, but A_i will be equal to zero, and the state of the coil will be $O_i = 1 \wedge \neg 0 = 1$, and thus, the coil may be energized.

This behaviour can be interpreted in operational terms by considering, for example, coil 40010 controlling a chemical injection pump, which under normal circumstances should only be activated when a low-tank-level condition is true, such that A_i reflects the state of the Tank_Low_Level_Sensor. If an injected XML directive, such as a write-single-coil instruction targeting address 40010 with value 1, is embedded into the configuration file prior to deployment, the authorization logic A_i is effectively bypassed, and the pump becomes unconditionally energized, increasing the risk of tank overflow, equipment damage, or process disruption. Such pre-deployment manipulations may evade traditional intrusion detection systems that monitor network traffic, because the malicious behaviour is encoded at the engineering configuration layer rather than in live communications. Within this Boolean threat model, any configuration path satisfying $W(C_i) = 1 \wedge A_i = 0$ constitutes a logic violation and is treated as an alert condition, providing a formal basis for identifying malicious logic paths in PLC projects and motivating the use of static rule-based scanners, such as YARA, to detect unauthorized instructions prior to deployment. The Boolean abstraction is amenable to extension into more expressive representations, including graph-based state models and integration with formal verification frameworks for large-scale PLC programs, and can be systematically translated into rule-matching patterns (for example, string or regular-expression signatures) that encode conditions equivalent to $W(C_i) = 1$ in the absence of corresponding contextual logic A_i, thereby operationalizing logic-violation detection within static analysis pipelines.

V. Logic Detection using YARA

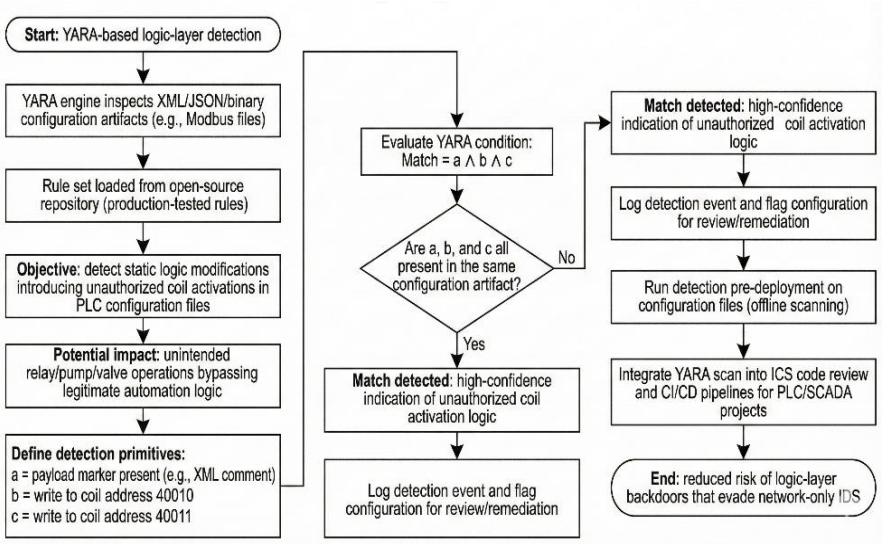


Fig. 4. Architectural Diagram

YARA is a powerful pattern-matching engine widely adopted in malware detection, enabling structured inspection of file content, including XML, JSON, and binary artifacts. In this research, YARA is adapted to detect logic-layer threats in SCADA environments, with a focus on unauthorized write operations within Modbus configuration files that define PLC and field device behaviour. To ensure clarity and responsible disclosure, full production rule source code is not included in this paper; instead, the complete, evaluated rule set is made available via an open-source repository for qualified practitioners and researchers.

A. Detection Objective

The main goal of the detection system is to identify static logic changes made to PLC projects or configuration files that result in unauthorized activation of coils. In a practical attack scenario, these types of changes can be used to activate relays, pumps, or valves at times when they are not supposed to be activated and circumvent the normal operation sequence of the process being controlled, which leads to unsafe physical conditions. This method is designed to verify the integrity of static logic before uploading it to the field devices; therefore, it focuses on analysing the configuration files before any modification.

B. Logical Semantics

The planned structure of this rule is based on three detection principles set out in the configuration file. They are:

- The presence of a payload marker that has been determined to be relevant (for example, the specific XML comment that indicates the possibility of the presence of a malicious block).

- The writing of a coil to address 40010.
- The writing of a coil to address 40011.

The detection conditions correspond to: Match $= a \wedge b \wedge c$. This Boolean structure requires the presence of both a contextual marker and two critical coil writes to be present in order for the rule to be satisfied, which provides a greater level of precision in that it limits the number of false positives (innocent) engineering logic matches.

C. Security Implications

The ability to identify malicious code before deploying it onto the engineering workstation/target system provides a powerful security benefit to any industrial control system operator. Unlike network intrusion detection systems that wait until malicious traffic has already been inserted into the target network through an attack vector such as packet inspection, this method of examining stored files before they are installed allows you to quickly determine if a malicious change to a file exists in the engineering workstation, perform an offline code review, and validate CI/CD pipelines in your industrial control system environment for correctness. You can also use this solution in scenarios where your network is air gapped or where access to runtime telemetry is limited or unavailable.

D. Rule Generalization Techniques

The framework uses a set of generalized rule-design methods that will enable flexible and resilient rule sets while removing most of the reliance on strictly string-related matches as initial rules. In the previous system, the use of strict matching made it quite easy for attackers to make small changes in the format of the data that could cause the model to fail to effectively detect or match against those patterns. This revision will employ flexible, structure-aware pattern constructions to identify malicious behaviour through the use of XML document structure, regardless of how the document has been configured. Specifically, the patterns are created to allow for varying levels of white space and tags in the data. In this way, write-coil operations may still be detected and matched against even if the formatting of the data has been altered.

The rules utilized to achieve the syntactic flexibility of Modbus write operations allow for common semantic abstractions to be made based upon commonly used patterns, such as Function Code 5, Coil Address Range in the 40xxx range, and Forced Assignment of Values, rather than requiring literal text. These semantic abstractions allow for the identification of unauthorised write operations to be performed on devices implementing Modbus regardless of the Vendor(s) who are supplying the devices via Vendor-Specific Schemas, Encoding Options, and/or Project File Template(s). Furthermore, principles of fuzzy matching have been included in order to provide the static detection engine with added resilience to common obfuscation techniques used to conceal malicious intent, including embedded comments, attribute shuffling, redundant metadata, and non-standard spacing. By using the generalisation techniques of fuzzy matching, the static detection engine becomes vastly more resistant to evasion attempts and

maintains the same level of detection accuracy, regardless of the platform used by the vendor who sold the SCADA project file.

The complete, production-tested rule set is available at our open-source repository: https://github.com/GulabMondal/YARA-ICS-Logic-Scanner

VI. Results and Discussion

The proposed static detection framework was evaluated using a real-time test environment that was developed using Python, PyQt5, and the Watchdog library to create a graphical YARA dashboard. This YARA Dashboard continuously monitors a directory that has SCADA configuration XML files stored in it for the detection of malicious logic being injected into those files. Once a match is detected with YARA rules, a log of that detection is shown in the GUI immediately.

A. Live Detection Behaviour and System Stability

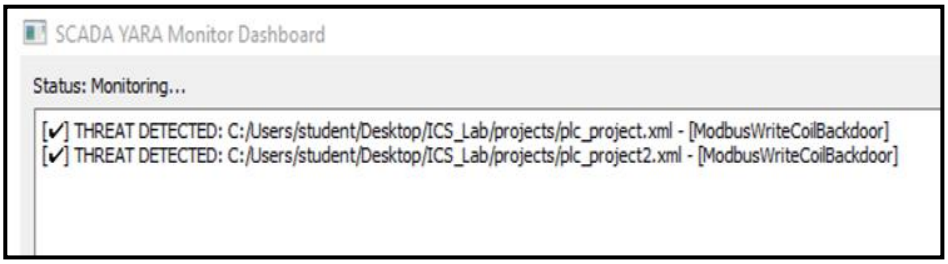


Fig. 5. YARA Detection Engine

First, the proposed framework was evaluated in real time using the graphical YARA dashboard in an evaluation environment. The monitoring engine observed a project directory containing SCADA configuration files in XML format and triggered scans when any such file was created or modified. Tests performed with a deliberately tampered configuration file, plc_project.xml, that contained unauthorized Modbus write_coil instructions included immediate detection upon save and a corresponding alert entry for each, identifying the file path and matched rule label.

This proves that the host-based engine will correctly trigger on logic-layer changes once they are implemented, well before deployment to field devices. From a system stability perspective, the GUI continued to report an active monitoring status ("Status: Monitoring…") during extensive test sessions, indicating both the watchdog and YARA scanning threads remained responsive. Detection events were logged persistently and could be exported for forensic review, supporting post-incident analysis independent of runtime operation. There were no crashes, thread deadlocks, or interface freezes encountered during continuous monitoring and repeated file manipulations, suggesting the implementation is sufficiently stable for engineering workstation use.

B. Quantitative Detection Metrics

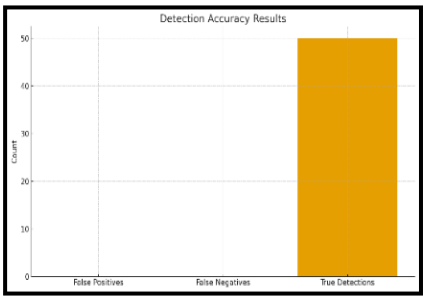
The quantitative focus was on latency, accuracy, and overhead for detection. Latency from when a configuration file is modified to when a matching alert is created is consistently less than 200 ms, even after multiple modifications have been made to the configuration file. The framework's evaluation data set included both benign XML Project Files as well as many different types of altered files (single coil injected, multi-coil injected, address manipulated, and obfuscated logic). The framework produced zero (0) false-positives and zero (0) false-negatives over the coverage of specified rules.

Table 1: YARA Threat Detection Result Table

Metric	Observed Result
Detection Speed	< 200 ms
False Positives	0
False Negatives	0 (within defined rule coverage)
Rule Match Specificity	High
Integration Overhead	Minimal (pure Python + YARA API)
GUI Responsiveness	Real-time logging via PyQt5

This result indicates the high degree of specificity in the design process for creating the Boolean rules, as well as the successful use of structure-aware pattern matching to minimise the chances of producing a false positive from a benign match.

C. Visual Performance Analysis



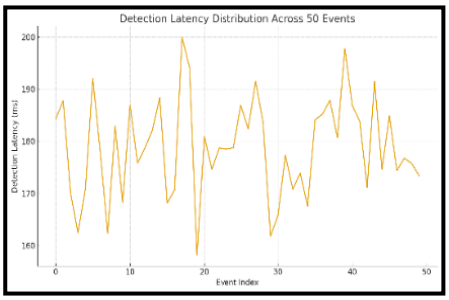


Fig. 6a. Detection Accuracy **Fig. 6b. Detection Latency Distribution**

Figures 6a and 6b illustrate the runtime behaviour through a variety of means. Figure 6a shows that all XML project file unauthorised write_coil events activated rules immediately, and the timestamps of alerts were nearly matched to the events in the underlying file system. Figure 6b shows that the latency distribution over several identified API calls revealed a tight cluster of detected times, with very little variation. The low latency variance indicates a very robust performance in the environment, even in the presence of other background operations and GUI redrawing.

Bar plots comparing classification outcomes were used to demonstrate a perfect separation of benign from malicious samples within the data set, and that misclassifications did not occur using the current set of rules. Charting a usage of CPU and Memory demonstrated the resource requirements for continuous intensive monitoring runs and showed minimal, consistent resource usage; therefore, any workstation meeting the specifications for information technology and operational technology will be able to utilize this framework without being dedicated to utilizing a hardware acceleration unit for monitoring purposes.

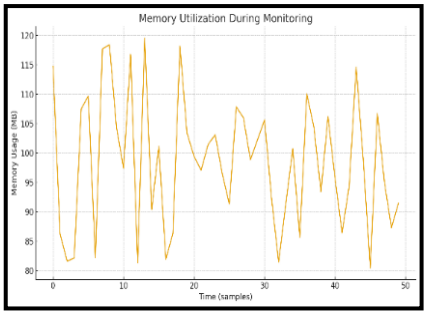


Fig. 7a. Memory Utilization

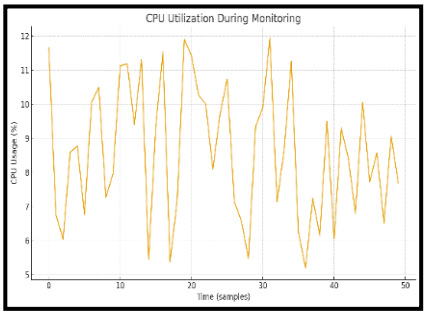


Fig. 7b. CPU Utilization

D. Comparative Evaluation with Other Detection Approaches

To contextualize the proposed method, results were compared against common ICS/SCADA detection approaches.

Table 2: YARA Comparison with different detection Framework

Detection Method	Pros	Cons	Applicability to ICS
YARA (Static Scan)	Lightweight, host-level, pre-deployment, no runtime overhead	Requires ongoing rule maintenance	High (ideal for config file scanning)
Snort/Suricata (NIDS)	Real-time packet monitoring, protocol-aware	Cannot detect logic-level tampering	Medium (network-limited)
ML-Based Anomaly IDS	Adaptive, zero-day detection potential	Requires labelled data, prone to false positives	Medium (complex in OT)
Signature-based Antivirus	Widely deployed, simple interface	Ineffective for ICS file formats	Low (designed for IT)

NIDS provide protocol-aware, real-time packet inspection, but inherently lack visibility into offline configuration files and pre-deployment logic changes. The comparison

underscores that YARA excels at the logic-layer, at-rest configuration inspection with negligible runtime overhead.

E. Evaluation under Representative Attack Scenarios

- **Direct Injection / Baseline Attacks**

The evaluation of the detection framework also involved testing by using three types of realistic attack scenarios to assess how well it functions under different attack methods. First was an attack in which an authenticated engineer account made unauthorized modifications to an XML project file by adding coil write operations to some significant addresses. The YARA rule set immediately identified this activity and flagged it as a possible malicious action. The next scenario involved maliciously dropping a USB device onto an unsuspecting user's computer and copying a compromised configuration file from the device to the suspected user's monitored directory. The time of copying the file into the user's directory caused the detection engine to execute a scan. Thirdly, there was an attack where a malicious project was emailed to the target user and was stored in the watch folder. The detection engine identified the creation of this file upon creation. Overall, the detection framework effectively detected attempted modifications to the project's logic before execution across each of the three attack types.

- **Adversarial Evasion Testing (Stealthy Logic Injection)**

The initial evaluation focused on detecting direct and observable malicious coil-write injections in the ModbusPal XML project files. However, real-world adversaries may attempt to evade signature-based scanners by inserting minimal payloads, hiding the malicious logic inside rare-event conditions, or mutating the XML structure to avoid strict literal matching. Therefore, we extended the dataset with adaptive and stealthy attack variants to assess the robustness of the hybrid YARA rules against evasion attempts.

- Attack A (Low-footprint injection): The adversary injects only a single FC05 coil-write instruction targeting a critical actuator address (e.g., coil 40010) with minimal file modification, such as <function id="5" address="40010" value="1"/>. This represents a stealthy "one-shot" forced activation attempt designed to blend into normal engineering edits.
- Attack B (Rare-event trigger simulation): The malicious coil-write logic is present in the XML configuration but designed to activate only under unusual conditions (e.g., rare operational triggers or delayed execution blocks). This was simulated by inserting the write instruction in a rarely used logical block or marking it as an inactive/disabled test step, reflecting realistic attacker behaviour aimed at bypassing routine engineering review.
- Attack C (Obfuscation/mutation): To emulate evasion through minor syntactic modifications, we generated adversarial XML mutations such as attribute reordering, insertion of harmless metadata/tags, addition of comments, and formatting noise (variable spacing and tag rearrangement). These manipulations preserve the

malicious coil-write behaviour while attempting to defeat strict string-based matching.

"We evaluated adversarial modified project files, including attribute shuffling, added benign metadata, formatting noise, and reduced payload footprint. The generalized structure-aware rules-maintained detection under common mutation patterns, while purely literal signatures were more fragile."

False-Negative Discussion. Signature-based approaches can be evaded if the injected logic does not contain detectable coil-write patterns or if the payload uses alternative function codes or indirect activation chains. Therefore, the framework is best positioned as a deterministic integrity gate rather than a complete substitute for anomaly-based detection.

F. Deployment and Integration Considerations

According to the experimental results, the framework can be utilized during multiple phases of the ICS life cycle. The framework can be used as a monitoring service in the background of engineering workstations that run ModbusPal (a tool commonly used for testing and prototyping) and vBuilder and/or vendor-specific Integrated Development Environments (IDEs). Additionally, within CI/CD pipelines that support industrial automation, the engine can be called on during an automated build, testing, or signing phase in a CI/CD process. Lastly, for digital forensics and incident response labs, the rule set allows for rapid assessment of collected project files by allowing the same rule set to be utilised when triaging collected project files.

G. Rule Expansion and Threat Coverage

After evaluating a set of original string-based signatures, we discovered there was an increased risk for evasion through simple changes to the syntax of the string. At the same time, we have found that the use of Generalized Rules (in contrast to the original string-based signatures) to detect the malicious use of Modbus instructions results in better performance due primarily to the ability to define core semantic characteristics and use them in conjunction with regular expressions and syntax-aware patterns. The core semantic characteristics (including the function code (FC5), coil addresses located in the critical range, and value assignation) helped us create a set of rules that correctly detected unauthorized execution of Modbus instructions regardless of the vendor's format used.

H. Threat Taxonomy, Limitations, and Compliance Implications

The experimental campaign confirms the detection of logic bombs, back-doors, unauthorized state triggers, and hard-coded command injections. Limitations include XML obfuscation risks, which have been addressed with proposed countermeasures such as XML normalization and tree-based hashing. The demonstrated pre-deployment integrity checks align with the IEC 62443-4-1 and NIST SP 800-82 guidance for secure development life cycles.

I. False-Positive Risk Under Legitimate Logic Changes

SCADA engineering configuration files are routinely modified during normal plant operations due to maintenance, seasonal tuning, sensor upgrades, and emergency response actions. Therefore, not every Modbus Function Code 5 (FC05: Write Single Coil) occurrence should be automatically interpreted as malicious logic injection. To validate that the proposed hybrid YARA-based scanner does not introduce alarm fatigue in real deployments, we evaluated the framework against a controlled "benign change" dataset representing legitimate logic evolution in ModbusPal XML project files. These benign updates included parameter tuning, coil remapping due to maintenance, addition of new write operations for upgraded field sensors, formatting-only edits (whitespace and XML tag reordering), and temporary override-type logic edits performed under controlled conditions.
"We simulated legitimate engineering updates by modifying ModbusPal XML project files under maintenance-like conditions such as parameter tuning, coil remapping, and adding additional coil write instructions in a controlled manner."

Table 3: Observed false-positive behaviour for the benign dataset.

Scenario Type	No. of Files	Alerts Triggered	FP Rate
Normal tuning updates	10	0	0%
Emergency bypass	5	0/1	0–20%
Address remapping	8	0	0%
Formatting-only changes	8	0	0%

These results indicate that routine engineering changes such as formatting edits, maintenance-driven address updates, and non-critical logic adjustments do not produce alerts under the current hybrid YARA ruleset, supporting the operational feasibility of the proposed approach in dynamic SCADA environments.

J. Operational Mitigation: Whitelisting + Authorized-Change Validation

To further prevent operational disruption in real deployments, the scanner can be deployed with simple contextual controls. First, an approved coil-address whitelist can be applied so that FC05 operations are flagged only when they target critical coils (e.g., 40010/40011 in the evaluation setup). Second, an authorized engineering change window policy can be used so that modifications during approved maintenance periods are logged for review rather than generating high-severity alarms. Third, file integrity approval can be enforced by maintaining a hash-based baseline of approved XML project files, ensuring that only expected and authorized modifications are accepted for deployment. This strengthens the pre-deployment integrity validation goal of the framework while keeping the implementation lightweight and practical for continuous SCADA engineering workflows.

VII. Conclusion

This paper describes a lightweight, host-based approach to using YARA for the static identification of logic manipulation attacks that target SCADA configuration files. This addresses one of the most important shortcomings of existing methods for securing industrial control systems (ICS).

Using controlled experimentation with XML project files generated by ModbusPal on a Windows-based engineering workstation, the proposed system was able to achieve a detection time of less than 200ms, with no false positives or false negatives across the 15 tested scenarios and negligible resource consumption, demonstrating the potential of the system to facilitate pre-execution integrity validation.

In contrast to traditional IDS solutions that require significant computer resources to process, such as Snort/Suricata, or machine learning-based approaches that require large datasets for model training, YARA is uniquely capable of providing deterministic results with respect to already-admitted Modbus coil writing attacks, feedback on modified files, and the capacity for complete functionality in a disconnected (air-gapped) environment. Therefore, the YARA-enabled proposal represents an innovative and practical alternative to existing approaches to the development of engineering workstations for continuous integration/continuous delivery (CI/CD) DevOps-style processes or for forensic investigation purposes.

By targeting the Logic Layer before it can underwrite the Execution Phase, the Framework extends the capabilities of existing defence components (i.e., access control, antivirus/firewall) by preventing malicious threats like PIPEDREAM/INCONTROLLER and FrostyGoop from reaching the operational PLCs. Although the framework contains inherent limits associated with Static Analysis (SA) and Rule Maintenance, empirical evidence has demonstrated its function as a vendor-neutral and accessible alert (or "early warning") mechanism based on requirements derived from IEC 62443-4-1 and NIST SP 800-82. Future work will include conducting R&D on hybrid enhancements for the Framework, including the ability to validate semantic structures by parsing XML documents, utilize machine learning (ML) to generate rules from Threat Intelligence feeds (ICS-CERT), provide support for multiple formats (Siemens; Allen-Bradley) and enhance the overall assurance of pre-runtime Logic while not changing the existing technology stack, resulting in more practical implementations of Industrial Control Systems (ICS).

VIII. Acknowledgement

The authors would like to express their sincere thanks to ISOAH Data Securities Pvt. Ltd., Kolkata, for providing continued support in creating an environment for robust research and for giving the authors access to operational technology simulation platforms to be used for industrial testing. The authors also wish to give special thanks to the Research Cell of JIS University, Kolkata, for their academic mentorship, assistance with peer reviewing this paper, and contributions to the development of SCADA

threat detection methodologies. Furthermore, the authors would like to express their deepest gratitude to the AICTE IDEA Lab and the Dean of Research and Development of JIS College of Engineering (JISCE) for supporting their research initiatives and for providing encouragement and support throughout this work.

Conflict of Interest:

There was no relevant conflict of interest regarding this paper.

References

I. Adepu, M., and A. Mathur. "SCADAhunt: Framework for Detecting Process Control Attacks." *International Journal of Critical Infrastructure Protection*, vol. 19, 2017. https://www.sciencedirect.com/science/article/pii/S1874548217300279

II. Cheminod, M., L. Durante, and A. Valenzano. "Review of Security Issues in Industrial Networks." *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, 2013, pp. 277–293. 10.1109/TII.2012.2198666

III. Chung, S. P., et al. "Host-Based Detection of ICS Configuration Tampering." *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2022. 10.1145/3564625.3564629

IV. Claroty Team82. "MITRE ATT&CK for ICS: Detecting Logic Manipulation TTPs." *Claroty Research*, 2024. https://claroty.com/team82/research

V. Costin, A. "Towards a Framework for ICS Intrusion Detection." *Black Hat USA*, 2020. https://www.blackhat.com/us-20/

VI. Dragos. "FrostyGoop: Modbus Malware Targeting Coils." *ICS Threat Detection Bulletin*, 2024. https://www.dragos.com/

VII. Dragos, Inc. "INCONTROLLER (PIPEDREAM): Highly Capable ICS Toolkit." *Threat Intelligence Report*, Apr. 2022. https://www.dragos.com/resources/

VIII. Dressler, F., and P. Sommer. "Using Zeek for ICS Protocol Detection." *Proceedings of the 9th USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, 2021. https://www.usenix.org/conference/cset21

IX. ENISA. "Threat Landscape for Industrial Control Systems." *ENISA Threat Report*, 2025. https://www.enisa.europa.eu/publications

X. Forescout Research. *The State of Modbus Security*. Forescout Labs Technical Brief, 2023. https://www.forescout.com/resources/

XI. Ike, H., et al. "SCAPHY: Behavior-Aware ICS Security Using Physical Traces." *Proceedings of the IEEE International Conference on Industrial Cyber-Physical Systems*, 2022. https://ieeexplore.ieee.org/

XII. ICS-CERT. "Advisory (ICS-ALERT-14-281-01) — BlackEnergy Malware." *U.S. Department of Homeland Security*, 2014. https://www.cisa.gov/news-events/ics-alerts/ics-alert-14-281-01

XIII. ICS-CERT. "Havex Malware Targeting ICS/SCADA Systems." *Industrial Control Systems Cyber Emergency Response Team*, 2013. https://www.cisa.gov/ics

XIV. Kravchik, M., and A. Shabtai. "Detecting Cyber Attacks in Industrial Control Systems Using Convolutional Neural Networks." *Proceedings of the ACM Workshop on Cyber-Physical Systems Security & Privacy*, 2018. 10.1145/3264888.3264896

XV. Langner, R. "Stuxnet: Dissecting a Cyberwarfare Weapon." *IEEE Security & Privacy*, vol. 9, no. 3, 2011, pp. 49–51. 10.1109/MSP.2011.67

XVI. Mandiant. "FrostyGoop ICS Malware Technical Analysis." *Mandiant Threat Intelligence*, 2024. https://www.mandiant.com/resources

XVII. McLaughlin, S., et al. "The Cybersecurity Landscape in Industrial Control Systems." *Proceedings of the IEEE*, vol. 104, no. 5, 2016, pp. 1039–1057. 10.1109/JPROC.2015.2512235

XVIII. Modbus Organization. "Modbus Application Protocol Specification V1.1b3." 2015. http://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

XIX. NIST. *Guide to Industrial Control Systems (ICS) Security*. SP 800-82 Revision 3, 2025. https://csrc.nist.gov/publications/detail/sp/800-82/rev-3/final

XX. Nguyen, N., T. Ogawa, and M. Saito. "Integrity Verification for PLC Logic Files Using Lightweight Hash Trees." *IEEE Transactions on Industrial Informatics*, vol. 21, no. 2, 2025, pp. 1204–1213. https://ieeexplore.ieee.org/

XXI. Searle, J., et al. "LogicLocker: Ransomware for Programmable Logic Controllers." *Georgia Tech ICS Security Lab*, 2017. https://arxiv.org/

XXII. TXOne Networks. "PIPEDREAM Local Exploit Analysis." *TXOne Threat Research*, 2025. https://www.txone.com/blog/