Toward Agentic Intrusion Detection in the Internet of Things: Rule Generation and Live Validation for XRCE-DDS Attacks

Matheus M. Ciocca[1], **Emanuel C. Ferreira**[1], **Tuigg R. Barcelos**[1,2], **Silvio E. Quincozes**[1,2], **Diego Kreutz**[1,2], **Paulo Silas Severo de Souza**[1,2]

[1]AI Horizon Labs, Federal University of Pampa (UNIPAMPA)

[2]PPGES, Federal University of Pampa (UNIPAMPA)

{matheusciocca, emanuelferreira, tuiggbarcelos}.aluno@unipampa.edu.br

{silvioquincozes, paulosilas, diegokreutz}@unipampa.edu.br

Abstract. *Signature-based IDSs rely on rules that must accurately capture attack behavior, yet public repositories provide limited coverage for emerging IoT and cyber-physical protocols such as XRCE-DDS. We present Rules Farmer, an open-source agentic tool that generates, deploys, and adversarially validates Snort 3 rules. A Defense Agent synthesizes signatures from natural-language intent, while an Attack Agent executes Docker-based attacks and evasive variants. Across four XRCE-DDS scenarios, 167 executions and 142 variants, 3 of 4 initial rules failed against the canonical attacks and required 15–23 revisions per scenario to converge. All campaigns converged within 50 executions, showing that live adversarial validation can expose semantic weaknesses that static validation misses.*

1. Introduction

The rapid expansion of the Internet of Things (IoT) and cyber-physical systems has increased the exposure of specialized communication protocols to network-based attacks [Quincozes et al. 2024]. XRCE-DDS is particularly relevant because it is the default middleware for micro-ROS, the official ROS 2 extension for microcontrollers, and is also used as the default PX4–ROS 2 bridge, placing it in robotics, industrial automation, and drone applications [eProsima 2025]. Its security risks include malformed messages, fragmentation abuse, session manipulation, and denial-of-service attacks [Trend Micro Research 2022], whose detection depends on protocol-specific semantics and misuse patterns rather than simple byte signatures.

Signature-based Intrusion Detection Systems (IDSs), such as Snort 3, are practical for monitoring network traffic and generating alerts from explicit rules [Cisco Snort Team 2024]. Their effectiveness, however, depends on signatures that accurately capture the target behavior. Although public rule repositories provide broad coverage for conventional IT environments, support for emerging IoT protocols is limited [Cisco Talos Intelligence Group 2026]. This creates a practical gap: defenders may need protocol-specific signatures before attacks occur, yet developing them requires expertise in IDS syntax, protocol semantics, and adversarial behavior. Existing approaches automate parts of this process through passive datasets [Adewole et al. 2025, Husnain et al. 2022], LLM-based translation of threat descriptions [Meng et al. 2026, Lian et al. 2025], or reactive adaptation

after evasions are observed [Li et al. 2025, Fu and Williams 2026, Moreno et al. 2025]. However, most rely on syntactic checks, static validation, or reactive feedback, leaving a gap in validating whether a generated rule remains effective against adversarial traffic.

This paper presents *Rules Farmer*[1], a proof-of-concept *agentic* tool that generates, deploys, and dynamically validates Snort 3 rules against XRCE-DDS attacks. The evaluation covers four scenarios: UDP DoS, Malformed Inject, Session Hijack, and Fragment Abuse. The contributions are: (i) *Rules Farmer*, a closed-loop agentic tool for IDS rule generation and adversarial validation, released as open source with the attack scenarios and orchestration artifacts; (ii) an empirical evaluation on a Snort 3 testbed, including a syntax-only ablation and a taxonomy of mutation operators used by the Attack Agent; and (iii) evidence that live adversarial validation exposes semantic weaknesses that static validation can miss.

2. Related Work

Table 1 compares related work across target domains, threat models, evaluated scenarios, and validation methods. In IoT settings, rule induction and traffic-based detection have largely relied on passive datasets or captured traces [Adewole et al. 2025, Husnain et al. 2022, Singh et al. 2024], treating the traffic as fixed observations rather than as an adaptive adversarial process. More recent approaches use LLMs to translate security knowledge into executable rules. UniRule [Meng et al. 2026] uses Retrieval-Augmented Generation across multiple detection languages, RuleMaster+ [Lian et al. 2025] leverages PoC artifacts, GRIDAI [Li et al. 2025] explores agentic rule generation and repair, VibeWAF [Fu and Williams 2026] adapts WAF policies to observed evasions, and [Moreno et al. 2025] targets ICS and IoT environments.

Table 1. Comparison between related work and the proposed approach.

Reference	Domain	Threat Model	Scenarios	Validation
[Li et al. 2025]	Enterprise	Passive (PCAP)	50 attack types	Static
[Meng et al. 2026]	Enterprise	None (translation)	Multi-language	Theoret.
[Lian et al. 2025]	Enterprise	Passive (PoCs)	3002 PoC instr.	Static
[Fu and Williams 2026]	Web (WAF)	Reactive (logs)	12 categories	Reactive
[Moreno et al. 2025]	ICS/OT	Passive (packets)	3 scenarios	Static
[Adewole et al. 2025]	IoT	Passive (dataset)	33 (CICIoT2023)	Static
[Husnain et al. 2022]	IoT (MQTT)	Active (testbed)	MQTT DoS/DDoS	Live
[Singh et al. 2024]	IoT (SDN)	Passive (emulated)	Multiple IoT types	Emulation
This work	**IoT (XRCE-DDS)**	**Active (adversarial)**	**4 XRCE-DDS**	**Live**

Despite these advances, Table 1 reveals a validation gap. Existing approaches rely primarily on theoretical analysis, static checks against offline captures, emulated traffic without adaptive attacks, or reactive adaptation after evasions appear in logs. *Rules Farmer* instead closes the loop between rule generation and live adversarial validation: each generated rule is deployed in a Snort 3 testbed, exercised against Docker-based XRCE-DDS attacks and their variants, and iteratively refined based on the resulting IDS alerts.

3. Agentic Architecture and Validation Workflow

At a high level, *Rules Farmer* implements a closed-loop generation and adversarial validation process. A user submits a natural-language intent, and the Defense Agent translates

[1] `https://github.com/GT-IoTEdu/rules-farmer_sbseg26`
Video (installation and features): `https://youtu.be/RoT96EwONks`

it into a Snort 3 rule. The rule must first pass a benign-traffic gate before being deployed to the IDS. The Attack Agent then inspects the deployed rule and generates a variant designed to evade it. The Orchestrator monitors whether Snort 3 detects the variant and feeds the outcome back into the loop: successful rules are retained, whereas evaded rules trigger another generation cycle. The process stops after a rule withstands k consecutive adversarial rounds. Fig. 1 illustrates the workflow and numbers the steps described below.

The architecture comprises three components: an Orchestrator, a Defense Agent, and an Attack Agent. Both agents use the `deepseek-v4-pro` LLM [DeepSeek-AI 2026], selected for its reasoning and code-generation capabilities rather than simply its availability. The LLM backend is configurable independently for each agent, allowing Defense and Attack to use different models; in this proof-of-concept, both use the same model. The offensive scenarios and the complete emulated test environment are publicly available.

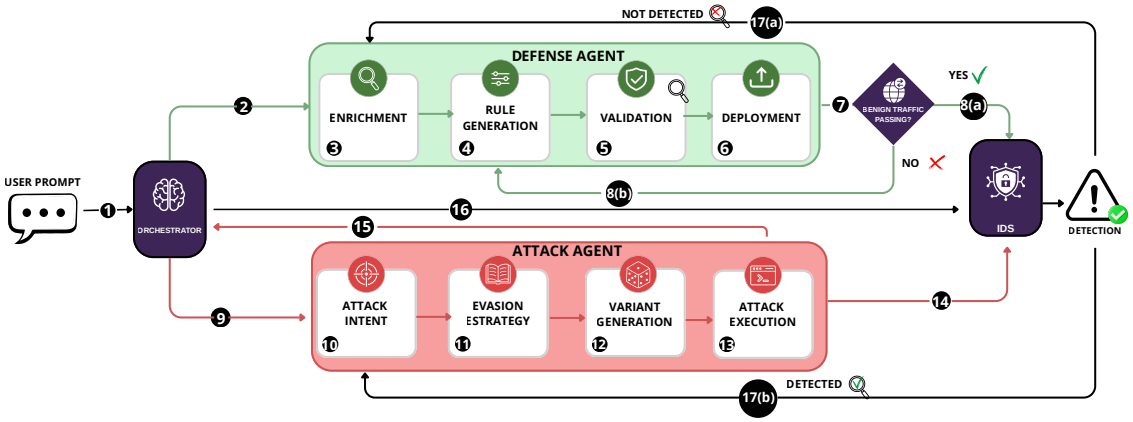

Figure 1. Proposed agentic architecture and workflow.

3.1. Orchestrator

The Orchestrator acts as the control plane. Upon receiving an intent (step 1), it maps it to an attack family and target coordinates (the IP address and port of the XRCE-DDS Agent), then manages the validation cycle. It invokes the Defense Agent (step 2) or Attack Agent (step 9) through an SSH control channel, collects execution outcomes (step 15), persists each iteration, and inspects IDS alerts (step 16) to determine the next action (steps 17(a) and 17(b)). Communication between the Orchestrator and agents uses structured JSON containing the intent, target coordinates, deployed rule, IDS verdict, and `evasion_rationale`, providing a consistent schema for prompts and feedback.

The outer loop runs for $N+1$ executions, with index 0 representing the base attack and subsequent executions representing attack variants. If no rule is deployed, the Defense Agent generates, validates, and deploys one (steps 2–8(a)). If the active rule detected the previous variant, it is retained and only a new variant is requested (step 17(b)). Each detected variant increments the consecutive-detection counter; an evasion resets it and sends the failure back to the Defense Agent (step 17(a)) for rule refinement or replacement. Whenever a new rule is generated, the base attack is re-executed to verify detection of the canonical case. Thus, the total number of executions exceeds the number of variants by the number of rule regenerations. A campaign ends as *converged*, *partial*,

or *failed*, with iteration artifacts stored in JSON/CSV and validated rules exported by attack family.

3.2. Defense and Attack Agents

The Defense Agent transforms an attack intent into a candidate rule through four stages: intent enrichment (step 3), rule synthesis with SID assignment (step 4), syntax validation (step 5), and deployment (step 6). The deployed rule then undergoes the benign-traffic gate described below before becoming active, after which the Orchestrator verifies its alerts (step 16). If the rule fails to detect the attack, the outcome is recorded as feedback for the next synthesis attempt.

Before any attack is executed, each candidate rule must pass a benign-traffic false-positive gate (step 7). The tool generates legitimate XRCE-DDS traffic toward the victim using three distinct profiles: a keepalive *ping*, a *participant-registration* message with a larger payload, and a *heartbeat*. Each profile runs at a low, client-like rate below the rules' volumetric thresholds. Varying operation, payload size, and timing helps expose rules that match legitimate traffic rather than a single fixed pattern. A rule that produces no alert is accepted and activated in the IDS (step 8(a)). If it triggers on benign traffic, its SID is identified in the alert log and the rule is rejected as over-generic and returned directly to synthesis for a narrower candidate (step 8(b)). Intent enrichment is skipped because the attack intent remains unchanged, and the alert log is cleared before the next attack test. The gate is structural rather than advisory: the attack stage cannot proceed until the rule passes it. It therefore filters over-generic rules during synthesis, but it is not a false-positive rate measurement; quantifying false-positive rates under sustained mixed workloads remains future work (Sec. 6).

The Attack Agent defines the offensive scenarios (step 10) and generates evasive variants. It operates in a white-box setting with respect to the IDS rule: the deployed rule is provided in full so that the agent can derive an evasion strategy (step 11)[2]. The agent then synthesizes a concrete variant (step 12), which is executed in a Docker container (step 13) against the monitored victim (step 14), and reports the outcome to the Orchestrator (step 15).

Rather than mutating captured packets, the agent modifies the scenario source code and rebuilds its container, ensuring that each variant is an executable attack program. The mutations fall into recurring categories: payload size, rate and timing, byte-level content, fragmentation layout, session and header fields, and source port. These operators are not hard-coded; they emerge from the generated evasion strategy and are recorded verbatim in `evasion_rationale` for each variant, making the process auditable (Tab. 5). Functional preservation is checked at two levels: the mutated scenario must build successfully, and its container exit status is recorded. A timeout is flagged as a potentially non-functional variant and reported separately rather than counted as an evasion. This provides an execution-level oracle, but does not constitute a semantic guarantee that the intended attack effect was successfully delivered.

[2]This setting does not model an external attacker with privileged access to the IDS. Instead, it treats the agent as an internal adversarial evaluator that stress-tests rules under worst-case knowledge.

3.3. Convergence Criterion and Scope

The loop terminates when a rule survives k consecutive adversarial rounds, with $k{=}20$ in our evaluation. Let p denote the probability that an independently generated variant evades the active rule. The probability of observing k consecutive detections is $(1-p)^k$; requiring this probability to be at most 0.05 gives $p_{\max} = 1 - 0.05^{1/k}$. For $k{=}20$, this yields $p_{\max} \approx 13.9\%$. Increasing k provides diminishing returns: $k{=}30$ lowers the bound to 9.5%, while achieving $p_{\max} < 5\%$ requires approximately 59 consecutive detections. Thus, k is a deliberate confidence–budget trade-off rather than an arbitrary threshold. This bound assumes independent variant generation, although variants are generated sequentially against the same rule, and it characterizes robustness only with respect to the Attack Agent's variant distribution.

Two scope boundaries are important. The orchestration, agents, and LLM calls run on the defender's monitoring infrastructure and never on constrained IoT endpoints; the resource-limited device is only the monitored victim. *Rules Farmer* is therefore a signature-development tool whose output can be deployed to production IDS sensors. In addition, the generated signatures operate at the packet and flow levels over UDP. Threats requiring long-horizon correlation across sessions are outside the current threat model.

4. Experimental Evaluation

The testbed comprises three roles: an Orchestrator, an attacker host, and a victim host running XRCE-DDS services and Snort 3. Experiments were conducted without continuous background traffic; legitimate traffic was used only for the per-rule acceptance gate described in Sec. 3.2. Each scenario was allocated a budget of 50 executions and a convergence threshold of 20 consecutive detections. All four scenarios converged before reaching the execution limit. Tab. 2 summarizes the resulting campaigns.

Table 2. Execution summary. Detection is cumulative over the whole campaign, search phase included; every converged rule detects the canonical attack and survives $k{=}20$ rounds. Nothing was discarded.

Scenario	Exec.	Variants	Rules	Detection	Evasion
UDP DoS	41	40	17	61.0%	39.0%
Malformed Inject	42	37	15	66.7%	33.3%
Session Hijack	41	33	17	61.0%	39.0%
Fragment Abuse	43	32	23	46.5%	53.5%
Total	**167**	**142**	**72**	—	—

Malformed Inject achieved the highest detection rate (66.7%), followed by UDP DoS and Session Hijack (61% each), while Fragment Abuse reached 46.5%. The lower performance on fragmentation-based attacks suggests that variations in traffic structure and payload characteristics are harder to capture with compact signatures. Figure 2 separates the search phase from the final converged run and shows the longer search required by Fragment Abuse. Because evasion is the complement of detection over the same executions, we report detection rates only.

Figure 2 shows the execution timeline of each campaign, including detections, evasions, rule regenerations, and the start of the final 20-round convergence run. Fragment Abuse required 23 distinct rules, compared with 15 for Malformed Inject, before reaching the convergence phase. Despite these differences, all four campaigns converged between executions 22 and 24, reflecting the stopping criterion more than the intrinsic difficulty

of the scenarios. With a budget of 50 executions and k=20, at most 30 executions are available for rule search; three campaigns consumed 21–23 of them. Thus, a slightly longer search would result in a *partial* campaign, meaning that the observed convergence also depends on the chosen execution budget.

Figure 2. Per-execution timeline. Green: detection; red: evasion; thin lines: rule regeneration; blue: start of the converging run.

4.1. Value of Live Validation

To isolate the contribution of the adversarial loop, we use the first rule generated in each campaign, after syntactic validation, as a syntax-only baseline and evaluate it against the canonical attack. This baseline detects only 1 of the 4 canonical attacks (UDP DoS), while the other three fail outright, showing that syntactic validity is a poor predictor of detection effectiveness. The adversarial loop also changes the underlying detection hypothesis (Tab. 3): for UDP DoS, the rule shifts from "small, high-rate packets" to "large, low-rate packets", a near inversion that would be difficult to derive from offline captures alone.

Table 3. Initial rule (accepted by syntax-only validation) versus the rule that converged after live adversarial validation. Only `msg` and `rev` are omitted. "Fired" indicates whether the rule detected the canonical base attack.

Scenario	Stage	Rule body	Fired
UDP DoS	Initial	`dsize:<30; detection_filter:track by_src, count 500, seconds 5; sid:9000157;`	yes
	Converged	`dsize:1100<>1300; detection_filter:track by_src, count 5, seconds 10; sid:9000182;`	—
Malformed Inject	Initial	`content:%x%x%x%x%n; detection_filter:track by_src, count 3, seconds 30; sid:9000261;`	no
	Converged	`content:|04|; detection_filter:track by_src, count 20, seconds 10; sid:9000282;`	—
Session Hijack	Initial	`pcre:/\x01\x00..\x00{4}/s; detection_filter:track by_src, count 5, seconds 30; sid:9000314;`	no
	Converged	`dsize:36<>40; detection_filter:track by_src, count 1, seconds 60; sid:9000355;`	—
Fragment Abuse	Initial	`content:|52 54 50 53|; content:|16|; detection_filter:track by_src, count 50, seconds 5; sid:9000227;`	no
	Converged	`dsize:30<>100; detection_filter:track by_src, count 5, seconds 30; sid:9000260;`	—

The converged rules reveal a systematic shift in the Snort primitives that survive adversarial pressure. Tab. 4 summarizes the primitives used across the 72 distinct rules generated in the four campaigns. Rate limiting is nearly universal, with 71 rules using a `detection_filter`. The choice between payload inspection and size-based matching is strongly scenario-dependent: Malformed Inject, whose semantics depend on corrupted header fields, uses `content` in 10 of 15 rules and `dsize` in only one, whereas UDP

DoS and Fragment Abuse show the opposite pattern, with `dsize` appearing in 13 of 17 and 13 of 23 rules, respectively. Session Hijack is the only scenario in which the agent uses `byte_test` to inspect fields at fixed offsets.

Notably, three of the four converged rules abandon payload inspection in favor of size ranges combined with rate thresholds. This produces broader signatures that are harder to evade but may also increase the risk of matching legitimate traffic. The integrated false-positive gate (Sec. 3.2) is designed to prevent such rules from being accepted during generation, while a quantitative false-positive study under sustained mixed workloads remains future work (Sec. 6).

Table 4. Detection primitives explored across the 72 distinct rules synthesized during the four campaigns. A rule may combine several primitives.

Scenario	Rules	`content`	`pcre`	`dsize`	`det._filter`	`byte_test`
UDP DoS	17	3	5	13	17	0
Malformed Inject	15	10	5	1	15	0
Session Hijack	17	7	4	7	16	5
Fragment Abuse	23	8	4	13	23	0
Total	**72**	**28**	**18**	**34**	**71**	**5**

4.2. Mutation Operators and Functional Preservation

Classifying the `evasion_rationale` of all 142 variants yields Tab. 5. The distribution is consistent with the attack scenarios: payload-size and rate/timing manipulations dominate the volumetric attacks, byte-level modifications appear in 100% of Malformed Inject and 94% of Session Hijack variants, and fragmentation mutations occur in 100% of Fragment Abuse variants but nowhere else. The agent therefore targets the specific feature on which the active rule relies rather than perturbing traffic indiscriminately. Regarding functional preservation, 11 of 142 variants (7.7%) ended in container timeouts, concentrated in Malformed Inject and Fragment Abuse. These variants are treated as potentially non-functional and are excluded from successful evasions, making the reported evasion rates conservative.

Table 5. Mutation operators across the 142 variants (% of variants; a variant may combine operators).

Operator	UDP DoS	Malf. Inj.	Sess. Hij.	Frag. Ab.
Payload size	98%	86%	91%	100%
Rate / timing	98%	86%	67%	88%
Byte-level content	20%	100%	94%	25%
Fragmentation	0%	19%	0%	100%
Session / header	2%	95%	18%	3%
Source port	12%	8%	3%	19%
Variants	40	37	33	32

Overall, the results support the main hypothesis: agentic rule generation benefits from live adversarial validation because evasive variants expose semantic weaknesses that static validation would likely miss. Because both agents use the same LLM backend, however, part of the observed effect may be model-driven rather than architectural; evaluating different backends (Sec. 6) is needed to disentangle these factors. The objective is not to claim operational robustness, but to show that coupling rule synthesis with attack execution provides a more realistic evaluation of generated Snort signatures.

5. Availability, Reproducibility and Demonstration Plan

Rules Farmer is released as open source with the Orchestrator, both agents, runnable XRCE-DDS scenarios, Snort 3 configuration, validated rules, raw artifacts for all 167 executions, and `analysis.py`, which regenerates all tables and figures. A single host with Docker, Snort 3, Python 3.12, and an LLM API key is sufficient. The provided scripts support both minimal execution and full reproduction of the evaluation budget (1 base attack $+\,49$ variants, $k{=}20$), while the persisted artifacts allow all reported results to be reproduced without additional LLM calls.

Demonstration. The presenter runs the loop locally, showing rule synthesis, benign-traffic validation, adversarial mutation, container rebuild, and IDS feedback. For safe and self-contained execution, the attack is processed by Snort 3 in file mode rather than transmitted over the network; packet content and mutations remain unchanged. A recorded campaign can also be replayed from the artifacts without LLM calls. Only a projector and power outlet are required from the organizers; live synthesis additionally requires Internet access to the LLM endpoint.

Generative AI disclosure. Both agents are LLM-driven (Sec. 3), and generative AI assisted with language editing. The design, experiments, analysis, and conclusions are the authors' responsibility, and all reported results are derived from the persisted experimental artifacts.

6. Final Remarks and Future Work

This paper presented *Rules Farmer*, an open-source, proof-of-concept agentic tool for generating, deploying, and dynamically validating Snort 3 rules against XRCE-DDS attacks. By deploying each candidate rule in a real Snort 3 instance and subjecting it to live adversarial execution, the approach moves rule assessment beyond syntactic plausibility toward empirical evidence from IDS alerts. Across four scenarios, most syntactically valid initial rules failed to detect the canonical attack and required adversarial feedback to produce effective rules. The resulting revisions also exposed how changes in timing, payload structure, fragmentation, and session parameters can invalidate signatures that appear correct under static validation.

The study is limited to four XRCE-DDS scenarios, a single IDS engine, and a controlled testbed. Although the benign-traffic gate (Sec. 3.2) rejects over-generic rules during generation, its outcomes are not currently persisted, preventing a quantitative false-positive analysis under sustained mixed workloads. Generation latency, computational overhead, and LLM cost are also not evaluated. Future work will quantify these operational aspects, extend the evaluation to additional IoT protocols and stateful multi-stage attacks, use distinct LLM backends for the Defense and Attack Agents to isolate model effects, strengthen functional-preservation checks with protocol-aware oracles, and incorporate continuously updated cyber threat intelligence. These extensions will help determine whether the benefits of live adversarial validation persist across broader protocols, threats, and deployment conditions.

Acknowledgments: This work was supported by the Hackers do Bem Program, executed by RNP and SENAI, coordinated by Softex, and funded by MCTI through the ICT Law (Law No. 8,248/1991), with additional support from CAPES (Código de Financiamento

001), CNPq (Grant No. 409743/2025-9), and FAPERGS (Grant Nos. 24/2551-0001368-7 and 24/2551-0000726-1). The authors also acknowledge the use of Generative AI to support code and manuscript review and improvement.

References

Adewole, K. S., Jacobsson, A., and Davidsson, P. (2025). Intrusion detection framework for internet of things with rule induction for model explanation. *Sensors*, 25(6).

Cisco Snort Team (2024). Snort 3 rule writing guide. Accessed: May 15, 2026.

Cisco Talos Intelligence Group (2026). Snort 3 community ruleset. Accessed: May 21, 2026.

DeepSeek-AI (2026). DeepSeek-V4-Pro: Model card and api documentation. Model identifier `deepseek-v4-pro`. Accessed: May 15, 2026.

eProsima (2025). Micro XRCE-DDS documentation. Accessed: May 15, 2026.

Fu, Q. and Williams, D. (2026). Toward LLM-driven rule generation for enforcement systems: An exploratory study on WAF. AgenticOS Workshop, arXiv preprint. Accessed: May 15, 2026.

Husnain, M., Hayat, K., Cambiaso, E., Fayyaz, U. U., Mongelli, M., Akram, H., Ghazanfar Abbas, S., and Shah, G. A. (2022). Preventing mqtt vulnerabilities using iot-enabled intrusion detection system. *Sensors*, 22(2).

Li, J., Chai, Y., Du, L., Duan, C., Yan, H., and Gu, Z. (2025). Gridai: Generating and repairing intrusion detection rules via collaboration among multiple llm-based agents.

Lian, W., Zhang, C., Zhang, H., Jia, B., and Liu, B. (2025). Rulemaster+: Llm-based automated rule generation framework for intrusion detection systems. *Chinese Journal of Electronics*, 34(5):1402–1415.

Meng, C., Le, W., Li, X., Wang, Q., Ren, F., Jiang, Z., and Liu, B. (2026). From context to rules: Toward unified detection rule generation.

Moreno, M., Sáez-de Cámara, X., Urbieta, A., and Iturbe, M. (2025). Leveraging LLMs for automated IDS rule generation: A novel methodology for securing industrial environments. In *Proceedings of X Jornadas Nacionales de Investigación en Ciberseguridad (JNIC 2025)*, pages 113–120, Zaragoza, Spain.

Quincozes, V. E., Quincozes, S. E., Kazienko, J. F., Gama, S., Cheikhrouhou, O., and Koubaa, A. (2024). A survey on IoT application layer protocols, security challenges, and the role of explainable AI in IoT (XAIoT). *International Journal of Information Security*, 23:1975–2002.

Singh, A., Chouhan, P. K., and Aujla, G. S. (2024). Secureflow: Knowledge and data-driven ensemble for intrusion detection and dynamic rule configuration in software-defined iot environment. *Ad Hoc Networks*, 156:103404.

Trend Micro Research (2022). A security analysis of the data distribution service (DDS) protocol. Technical report, Trend Micro.