

APIOT: Autonomous Vulnerability Management Across Bare-Metal Industrial OT Networks

Adel ElZemity^{[0000-0002-5402-7837]1}, Budi Arief^{[0000-0002-1830-1587]1},
Shujun Li^{[0000-0001-5628-7328]1}, Calvin Brierley^{[0000-0001-8766-822X]1},
Yichao Wang^{[0000-0002-4633-3690]1}, Yuxiang Huang^{[0009-0001-5122-7710]2},
James Pope^{[0000-0003-2656-363X]2}, Haoxiang Li^{[0009-0002-3359-6464]2}, and
George Oikonomou^{[0000-0002-1684-6989]2}

¹ University of Kent, Canterbury, United Kingdom

{ae455,b.arief,s.j.li,c.brierley,yw301}@kent.ac.uk

² University of Bristol, United Kingdom

{ethan.huang,james.pope,haoxiang.li,g.oikonomou}@bristol.ac.uk

Abstract. Bare-metal operational technology (OT) devices – especially the microcontrollers running Modbus/TCP and CoAP at the base of industrial control systems – have remained outside the reach of autonomous security attacks. Prior autonomous pentesting studies target Linux and web systems, whose shells and filesystems are familiar to LLM agents. Bare-metal OT has neither, so agents must reason directly over protocol fields and parser semantics. This requires new action-space designs and runtime controls, and opens new research questions about protocol-level exploit reasoning and its deployment envelope. We present APIOT (Autonomous Purple-teaming for Industrial OT), the first large language model (LLM) framework demonstrating an autonomous attack and remediation of bare-metal OT devices, achieving the full discovery → exploitation → patching → verification cycle without step-by-step human intervention. We implemented and evaluated this framework on Zephyr RTOS firmware across heterogeneous industrial IoT (IIoT) topologies. Through 290 experiment runs spanning five frontier LLMs, three network topologies, two impairment levels, and guided versus unguided conditions, APIOT achieved a mission success rate of 90.0% on the full attack-remediation cycle. We found that the runtime governance layer (which we call an overseer) is a critical engineering variable: without it, agents exhibit systematic degenerate patterns, including repetition loops, missing crash verification, and reconnaissance deadlocks. Together, these findings carry two implications beyond our testbed. Attacker expertise is no longer the binding constraint on bare-metal OT exploitation, and defender threat models must now assume LLM-augmented adversaries capable of executing autonomous discovery-through-remediation cycles against industrial firmware. Moreover, the overseer can improve outcomes by structurally blocking bad sequences rather than changing agent behaviour: runtime governance is a reliability lever for this class of LLM agents, not a model-specific tuning choice.

Keywords: OT security · IoT security · autonomous vulnerability management · LLM agents · bare-metal MCU · industrial protocols

1 Introduction

Operational technology (OT) devices, such as industrial controllers, field sensors, and smart meters, present a growing and under-protected attack surface [6,28]. Unlike IT infrastructure, these devices run bare-metal firmware on resource-constrained microcontrollers (MCUs) with no operating system, no general-purpose shell, and kilobytes rather than megabytes of RAM. They communicate via industrial protocols such as Modbus/TCP [21] and CoAP [27], which carry device-specific semantics that differ fundamentally from HTTP and SSH. Security testing of these systems today is a manual, expert-intensive process: a practitioner must understand the protocol, craft raw payloads, and interpret device responses. This creates both an engineering bottleneck (scarce expert labour) and a research gap on whether autonomous agents can perform protocol-level exploitation on bare-metal targets, where device diversity and the scarcity of industrial protocol documentation in LLM training corpora leave open whether frontier models can reason at the parser level without fine-tuning.

A first research question therefore concerns capability: can autonomous LLM agents perform protocol-level exploitation on bare-metal MCU targets? Recent research work has explored whether large language models (LLMs) can automate penetration testing. Researchers behind PentestGPT [10], HackingBuddyGPT [14], and AutoAttacker [31] have shown that LLM agents can navigate Linux privilege escalation and web application attacks with notable autonomy. However, all published evaluations targeted Linux or web systems. To our knowledge, no published study has demonstrated autonomous LLM exploitation on bare-metal MCU targets. This remains a difficult research problem because these targets provide no shell or filesystem abstractions, and successful operation requires protocol-level reasoning rather than system-level intuition.

A second research question concerns reliability. LLM agents in autonomous loops are known to exhibit degenerate behaviours: they repeat failed actions, get stuck in subgoal loops, and make inconsistent progress [19,32]. Prior work addresses this primarily through prompt engineering or agent-level retries. We ask whether an overseer runtime governance layer (rule-based guardrails with optional LLM-based advisory steering) is sufficient to achieve reliable mission completion without fine-tuning, and how large the performance loss is when the overseer is removed.

To address these questions, we designed APIOT (Autonomous Purple-teaming for Industrial OT), an LLM agent framework that completes the full discovery, exploitation, patching, and verification cycle on bare-metal OT targets without per step human intervention. APIOT gives the agent access to low-level protocol primitives rather than named exploits, requiring it to reason about protocol fields and parser behaviour, and includes an overseer runtime governance layer that enforces reliability through rule-based guardrails. We evaluated APIOT in controlled experiments across 290 runs, spanning five frontier LLMs, three network topologies, and two impairment levels. Our study isolates two design choices: protocol-primitive tooling (raw protocol fields rather than pre-packaged named

exploits) and the Overseer runtime governance layer. This setup lets us measure both capability and reliability directly. Our main contributions are:

1. **The first empirical demonstration on bare-metal MCU targets:** To our knowledge, APIOT is the first end-to-end autonomous LLM agent demonstration on bare-metal MCU targets running industrial protocols, achieving autonomous discovery → exploit → patch → verify cycles without per-step human intervention during mission execution across single and multi-hop OT network topologies. This demonstration was evaluated on Zephyr RTOS firmware in our experimental setup. A sensitivity study confirmed that the findings can generalise across five frontier LLMs with 85.3% overall success (128/150; 95% Wilson confidence interval: 78.8–90.1%).
2. **Oversight as a structural reliability mechanism:** To the best of our knowledge, this is the first controlled ablation study in autonomous bare-metal OT pentesting that isolates a runtime governance layer against prompt-only operation. The ablation showed that lightweight rule-based middleware is associated with substantially more reliable behaviour and efficiency. Overseer-ON runs achieved 100% mission success versus 90.0% without oversight, completing missions 20.5% faster. Across the two ablation conditions tested (overseer-ON and overseer-OFF), consistent crash verification cases occurred only in overseer-ON runs.
3. **A taxonomy of systematic failures and a deployment envelope:** Autonomous OT agent failures are not random: across all observed mission failures, we identified four systematic patterns (protocol confusion, stall-and-repeat loops, phase deadlock, and infrastructure failure). This paper introduces what is, to the best of our knowledge, the first taxonomy of systematic failures of autonomous bare-metal OT agents. Autonomous OT agent failures are not random: we identified four systematic failure patterns that collectively account for all observed mission failures, enabling more targeted countermeasures. In our experiments, the deployment boundary for autonomous CoAP operation appeared in topology T3 (Edge–Fog–Cloud [22]), where traffic traversed three network layers and multiple hops before reaching the target.

The rest of this paper is organised as follows. Section 2 reviews background and related work. Section 3 presents the APIOT architecture, and Section 4 details the experimental methodology. Section 5 reports empirical results, while Section 6 discusses the implications of these results. Section 7 concludes the paper. Appendix sections provide implementation and configuration details.

2 Background and Related Work

2.1 Bare-Metal MCU Security and Testbeds

Resource-constrained MCUs running real-time operating systems (RTOS) such as Zephyr [33] or FreeRTOS [25] form the lowest layer of the OT stack. A typical

Cortex-M3 device [2] has 64–256 KB of RAM, 256 KB of flash, and runs a single firmware image with no MMU, no process isolation, and no dynamic linking. This means exploitation has no shell to land in and no filesystem to pivot through: success is measured by crashing the device (denial of service) or writing to control registers (actuation attack).

CoAP (Constrained Application Protocol, RFC 7252 [27]) is a UDP-based request-response protocol designed for resource-constrained nodes. It supports confirmable (CON) and non-confirmable (NON) messages; option fields encode resource paths and metadata. Malformed option headers (delta/length overflow) can exhaust parser state on devices with no bounds checking [1,3].

Modbus/TCP maps the Modbus application layer over TCP port 502. It uses a Modbus Application Protocol (MBAP) header carrying a length field; devices trusting this field without validation are vulnerable to heap overflow [17] when the declared length far exceeds the actual payload.

2.2 Autonomous Penetration Testing and Agent Reliability

Deng et al. [10] used GPT-4 in a reasoning-generation-tool loop to guide human-driven penetration tests, achieving competitive CTF performance. Happe and Cito [14] demonstrated fully autonomous Linux privilege escalation with HackingBuddyGPT via tool-calling LLMs. Xu et al. [31] extended Happe and Cito’s work to network reconnaissance and lateral movement on Linux hosts with AutoAttacker. Happe et al. [15] systematically evaluated LLMs’ capabilities for privilege escalation across multiple models. Fang et al. [11] showed that GPT-4 agents could autonomously exploit 87% of one-day CVEs given the vulnerability descriptions, extending autonomous exploitation to real-world CVE databases.

All of the above studies evaluated Linux targets with full POSIX semantics. To the best of our knowledge, prior work has not considered bare-metal MCU targets, industrial protocols, or the two-phase red-then-blue workflow. Our work addresses this research gap.

Yao et al. [32] introduced the thought-action-observation loop as a standard LLM agent architecture. Liu et al. [19] showed that 29 API-based and open-sourced LLMs struggle with long-horizon task completion, exhibiting repetition and inconsistency in complex agentic settings. Recent work on LLM agent safety and security has proposed various monitoring strategies, including prompt-level instructions and secondary LLM judges [9,18,23,24,29]. This literature leaves open whether lightweight rule-based guardrails, optionally augmented by a secondary LLM for advisory steering rather than as the primary enforcement mechanism, are sufficient to prevent dominant failure modes.

Kim et al. [16] and Chen et al. [7] developed firmware emulation systems for Linux-based router images at scale. These systems focus on automated dynamic analysis of existing firmware, not security agent orchestration. Ghazanfar et al. [13] proposed IoT-Flock, an open-source framework for IoT traffic generation, and tested it using generated IoT protocol traffic, but they did not model bare-metal devices or adversarial agent workflows.

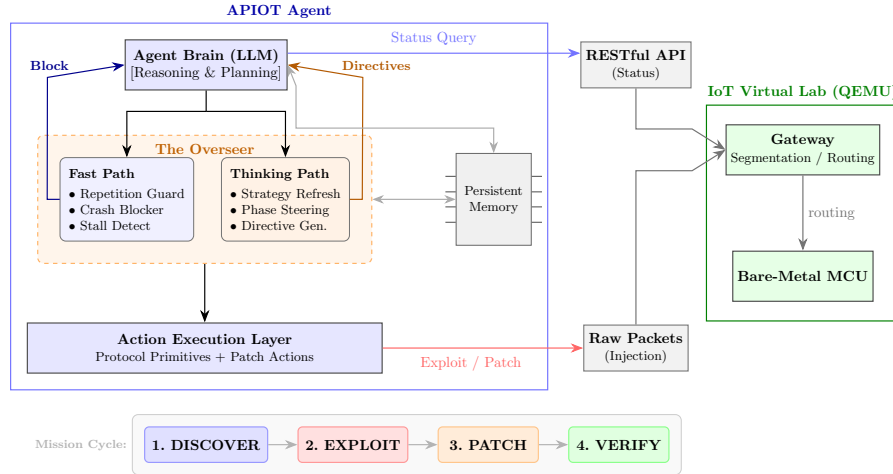


Fig. 1. High-level architecture of the APIOT autonomous purple-teaming framework and its interface with the IoT Virtual Lab testbed (see Section 3 for details).

For the discovery stage, the most directly comparable prior work on bare-metal MCUs is firmware fuzzing (automated generation of malformed inputs to discover crashes and memory corruptions in embedded firmware). Clements et al. [8], Scharnowski et al. [26], and Feng et al. [12] emulated MCU firmware and applied coverage-guided fuzzing to find crashes automatically; unlike our framework, this line of work does not address autonomous exploitation, patching, or verification stages. APIOT is complementary rather than competing: it operates as a reasoning agent (not a fuzzer), targets the full discovery \rightarrow exploitation \rightarrow patching \rightarrow verification cycle, and requires semantic understanding of *why* a payload triggers a parser failure rather than discovering it by exhaustive mutations. IoT Virtual Lab, our QEMU-based emulation testbed, differs in supporting Cortex-M3 bare-metal firmware alongside Linux devices on a shared network, with RESTful API control, multiple IIoT topology models, and configurable network realism, properties designed to support autonomous agent evaluation.

3 System Design

Figure 1 shows the overall architecture of our framework. APIOT is built around an Agent Brain, an LLM reasoning loop backed by a persistent memory store, that drives a four-stage mission cycle of discovery, exploitation, patching, and verification. An Overseer sits between the Agent Brain and the Action Execution Layer and enforces reliability through two paths: a rule-based fast path (repetition and crashed-target guards, phase-transition enforcement) and an optional LLM-based thinking path that provides advisory steering when the fast path detects a stall. IoT Virtual Lab emulates the target network, and APIOT

interacts with it solely through low-level raw packets and a read-only RESTful API for status queries; APIOT has no lifecycle control over IoT Virtual Lab, which ensures that measured capability reflects in-mission reasoning rather than orchestration privileges.

3.1 Design Objectives and Scope

The design of APIOT is driven by four requirements. **(R1) End-to-end autonomy:** the agent must complete discovery → exploitation → patching → verification within one mission, without per-step operator control. **(R2) Protocol-level reasoning:** the capability test should measure if the agent can infer exploit semantics from protocol structures, not retrieve canned exploit names. **(R3) Runtime reliability:** the architecture must prevent common long-horizon agent failures (repetition, stalled reconnaissance, and invalid phase progression). **(R4) Experimental isolation and reproducibility:** the target environment must be resettable, observable, and strictly separated from agent lifecycle control.

3.2 Core Design Decisions

Protocol primitives over named exploits. The action space exposes low-level protocol primitives rather than pre-packaged exploit labels. This forces the agent to reason about protocol fields and parser behaviours, so a success reflects protocol understanding rather than template matching.

Two-phase purple-team workflow. The mission is structured as Red (discover/exploit/confirm impact) followed by Blue (derive patch/verify mitigation). This decomposition aligns the objective with operational purple-team practice and makes failure attribution clearer than a monolithic task objective.

Runtime governance. Overseer places rule-based guardrails on tool-call sequences and mission progression, with optional LLM advisory steering only as a supplementary mechanism. This separates structural safety guarantees from model quality and enables direct ablation of the governance premise.

Strict environment boundary. The agent can observe topology and send packets, but it cannot spawn, reset, or terminate lab devices. This avoids hidden orchestration privileges and ensures measured capability comes from in-mission reasoning and action selection.

3.3 Instantiation for This Study

This conceptual design was instantiated and evaluated on IoT Virtual Lab across CoAP, Modbus/TCP, and MQTT conditions, with topologies T1–T3 (flat star, Purdue Model-segmented, and Edge–Fog–Cloud respectively). Implementation-level parameters (e.g., concrete tool roster, guard thresholds, and device/network configuration) are reported separately in Appendices A.1–A.3, while the evaluation protocol is defined in Section 4.

Table 1. Experimental conditions. T1 = flat star; T2 = Purdue Model (PERA)-segmented [30]; T3 = Edge-Fog-Cloud.

Dimension	Values
Protocol	CoAP (UDP/5683), Modbus/TCP (TCP/502), MQTT (TCP/1883)
Topology	T1, T2, T3
Overseer	Enabled, Disabled
Impairment	None; Medium (5% loss, 50 ms latency, 10 ms jitter); Heavy (20% loss, 200 ms latency, 40 ms jitter, HMI traffic)

4 Experimental Methodology

The evaluation is structured around our two key claims:

- **Claim 1 (capability and generalisability)** is assessed across protocols, topologies, impairment levels, and models: can the agent reliably complete the full attack-remediation cycle on bare-metal MCU targets, and can this hold across frontier LLMs and realistic network conditions?
- **Claim 2 (oversight as a reliability mechanism)** is assessed via a controlled ablation: can disabling the runtime governance layer produce more systematic behavioural failures, and can re-enabling it reduce or eliminate such failures?

Agent decision-making is characterised as a secondary evaluation goal: tool-call distributions and failure modes are logged to describe *how* the agent operates, independently of success/failure outcomes.

4.1 Experimental Conditions

Table 1 summarises the experimental conditions. All runs used a full purple-team mission (both red and blue phases mandatory). Each condition was replicated ten times; the run order was randomised. This design yielded 140 core runs (≈ 70 compute hours; ≈ 30 minutes/run), giving per-cell 95% Wilson confidence intervals with half-width below 15 percentage points near 100% success, which is sufficient to distinguish the ablation and topology effects reported in Section 5. The primary-condition runs fixed a single backbone model (MiniMax M2.5 via OpenRouter) to isolate protocol, topology, impairment, and oversight effects; cross-model robustness was evaluated separately in a 150-run model-sensitivity study. The sampling temperature was fixed at the default ($T = 0.7$) for all runs. Heavy impairment included background human-machine interface (HMI) traffic.

Claim 1 evaluation: 70 runs (CoAP/Modbus \times T1/T2/T3 \times 10 replicates + MQTT T1 \times 10). **Claim 2 ablation:** 30 new runs (overseer-OFF; CoAP/Modbus/MQTT \times T1 \times 10); the overseer-ON comparison arm reuses the corresponding 30 T1 runs from Claim 1. **Robustness:** 40 runs (2 impairment levels \times 2 protocols \times 10). **Generalisation (model sensitivity):** 150

Table 2. Primary evaluation metrics.

Metric	Definition	Claim
Mission success rate	Percentage of runs reaching <code>TASK_COMPLETE</code>	C1
Exploit success rate	Percentage of <code>execute_exploit</code> calls succeeding	C1
Patch verification rate	Percentage of patches confirmed by <code>verify_patch</code>	C1
Turns to first exploit	Turns before first successful exploit	C1, C2
Turns to completion	Total turns until <code>TASK_COMPLETE</code>	C1, C2
Redundant call rate	Repeated (tool, ip, args) / total calls	C2
Stall events	Turns with no new finding or patch	C2
Overseer intervention rate	Steering messages / total turns	C2
Time-to-remediate	Seconds: first exploit \rightarrow verified patch	C1

runs (five models \times three conditions \times ten replicates). **Total: 290 unique runs.**

Six targeted additions were made to support this evaluation, including an overseer-OFF flag, software MCU simulators, a parameterised experiment runner, enriched per-turn logging, analysis scripts, and impairment integration (details in Appendices A.1–A.3).

4.2 Metrics

Table 2 lists the primary metrics derived from the tool call log. For the global behavioural characterisation, we define four failure categories coded from session logs: **Protocol confusion** (wrong exploits for device type); **Stall-and-repeat** (identical (tool, ip, args) tuples called more than three times with no progress); **Phase deadlock** (red phase completed, blue phase never initiated despite open findings); and **Infrastructure failure** (device crashed mid-mission causing an inconsistent network state, ending in mission abort).

5 Results

The following results address the two key claims in turn. Sections 5.1 and 5.2 establish Claim 1 that protocol-primitive tooling makes bare-metal MCU exploitation tractable, and that the capability holds across models, topologies, and network conditions. Section 5.3 establishes Claim 2 that the runtime governance layer is a primary reliability mechanism, not a tuning choice.

5.1 The Threat was Demonstrated: Autonomous Exploitation Capability

Table 3 summarises the topology matrix outcomes for CoAP and Modbus (60 runs: T1/T2/T3 \times 10 replicates each). The results show an all-but-one pattern: APIOT achieved **100%** mission success in every matrix condition except CoAP

Table 3. Mission success rate by protocol and topology ($n = 10$ per cell) for CoAP and Modbus conditions. Median turns and mission duration are for successful runs only.

Protocol	Topology	Success	Median turns	Median duration
CoAP	T1	10/10 (100%)	11.0	3.0 min
CoAP	T2	10/10 (100%)	30.0	8.5 min
CoAP	T3	3/10 (30%)	18.0	5.0 min
Modbus	T1	10/10 (100%)	24.0	7.5 min
Modbus	T2	10/10 (100%)	23.0	4.5 min
Modbus	T3	10/10 (100%)	27.0	7.5 min
Overall		53/60 (88.3%)	–	–

on three-tier topology (T3), where it achieved 3/10 (30%). The additional MQTT baseline condition at T1 achieved 10/10 success; including it yields 63/70 (90.0%; 95% Wilson confidence interval: 80.8–95.1%) overall.

The seven failures occurred exclusively on CoAP T3, the three-tier Edge-Fog-Cloud topology where the MCU target sits behind two forwarding hops on the isolated OT segment. Session log analysis revealed the dominant failure mode as a **multi-target reasoning stall**: after crashing the first device in the OT cell, the agent generated three consecutive empty LLM responses when attempting to reason about lateral movement to the second hop, triggering the empty-response abort guard. The agent initiated the first exploit correctly within 3–4 turns in all failing runs; the breakdown occurs specifically at the multi-hop lateral movement step, not during initial target identification or exploitation. Modbus succeeded in all topologies including T3, where TCP’s stateful connection model provides implicit routing feedback that the stateless CoAP/UDP agent lacks.

Speed of threat identification. Across all 60 tabled runs, the agent initiated its first exploit attempt within 5 tool calls, regardless of topology or protocol. This represents a trivial recon-to-exploit gap: the agent reads the network state, identifies open ports, and launches a targeted exploit quickly with no human direction. Figure 2 shows the full timeline distribution.

Agent decision profile. Across 70 Claim 1 runs (60 CoAP/Modbus + 10 MQTT), the agent made 1,660 tool calls in total (mean 23.7 per run). Figure 3 shows the per-condition tool-call distribution for CoAP and Modbus; MQTT’s multi-step attack chain produced a distinct profile discussed in Section 5.3.

Exploit calls account for 27.0–51.0% of all calls across conditions (higher in Modbus, where target identification is faster). **Recon calls** account for 25.0–42.0%, with CoAP T3 showing the highest fraction (32.0%), consistent with the extra routing hops requiring more topology discovery. **Blue-team calls** account for 5.0–19.0%, a notably small fraction reflecting that remediation is concentrated in a short phase after exploit success.

Failure mode. The seven mission failures (CoAP T3) reflect a lateral-movement planning failure mode specific to stateless UDP protocols in multi-hop topologies, not a target-identification or exploit-generation failure. Zero runs ex-

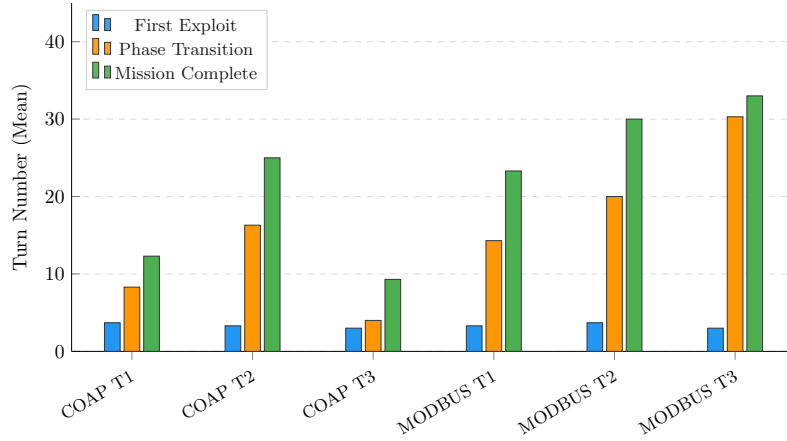


Fig. 2. Mean turn counts to three milestones: the first exploit attempt, the red-to-blue phase transition (attack complete, defence begins), and the mission completion. The recon-to-exploit gap (turns 3–4) is consistently short, and most of the mission is spent in the exploit-and-verify loop rather than the blue-team phase.

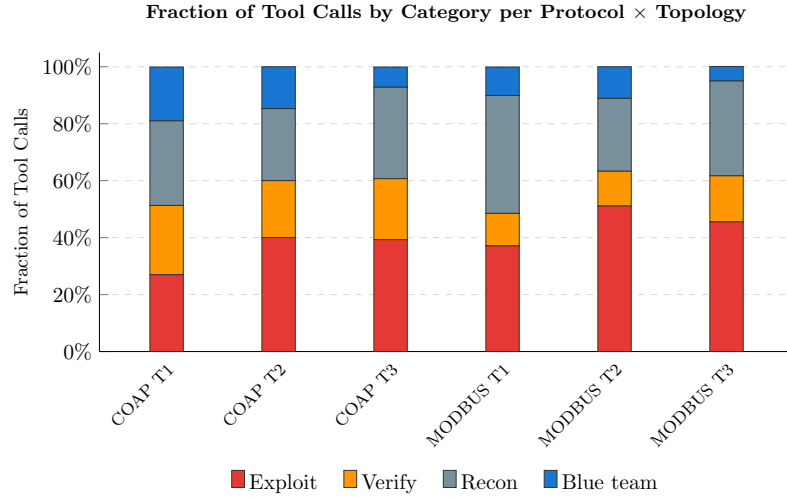


Fig. 3. Fraction of tool calls by category (exploit, verify, recon, blue team) per protocol × topology condition. Exploit and recon together dominate in all conditions; blue-team calls are consistently the smallest category (5.0–19.0%).

hibited the “zero exploits attempted” or “exploit succeeded, verify failed” failure modes, confirming that the agent’s protocol-specific exploit logic is reliable once

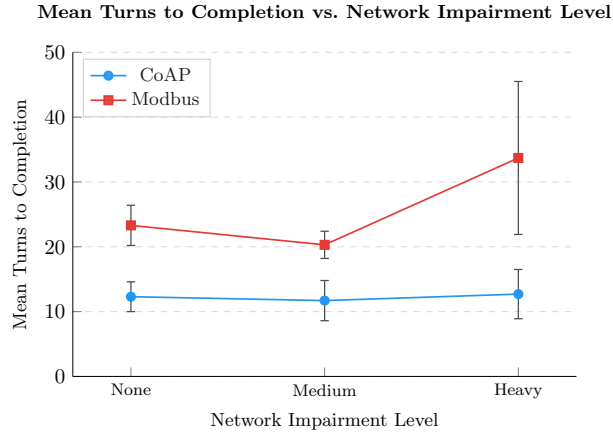


Fig. 4. Mean turns to mission completion vs. network impairment level (all runs successful at 100%). CoAP turn count is stable across all conditions (≈ 12 turns); Modbus increases from 23 turns at baseline to 34 turns under heavy impairment (20% loss, 200 ms latency), indicating efficiency degradation without mission failure.

it engages a target. All 63/70 successful runs also completed the blue-team phase, with the patch verification step confirmed in every case.

5.2 Capability Generalisability Across Topologies, Impairments, and Models

Figure 4 characterises the impairment aspect of the autonomous OT agent capability envelope.

Topology. Modbus/TCP achieved 100% success across all three topologies, including T3. CoAP/UDP succeeded at 100% on T1 and T2, but degraded to 30% on T3 via the lateral movement stall and TCP/UDP routing asymmetry described in Section 5.1. This establishes three-tier isolated OT cell topology as the current lateral-movement planning boundary for autonomous CoAP deployment, not a network transport limitation.

Network impairment. Both protocols proved robust across all tested impairment levels: CoAP and Modbus each achieved 100% mission success at medium (5% loss, 50 ms latency, 10 ms jitter) and heavy (20% loss, 200 ms latency, 40 ms jitter, HMI background traffic) conditions. CoAP’s UDP-based retransmission handles packet loss without connection state; Modbus/TCP tolerates the latency and jitter introduced without requiring connection re-establishment in these experiments. Network impairment is therefore not a deployment boundary within the tested parameter range.

Model sensitivity confirmed that the capability demonstrated in Section 5.1 is not an artefact of a single LLM choice. To address the single-model gener-

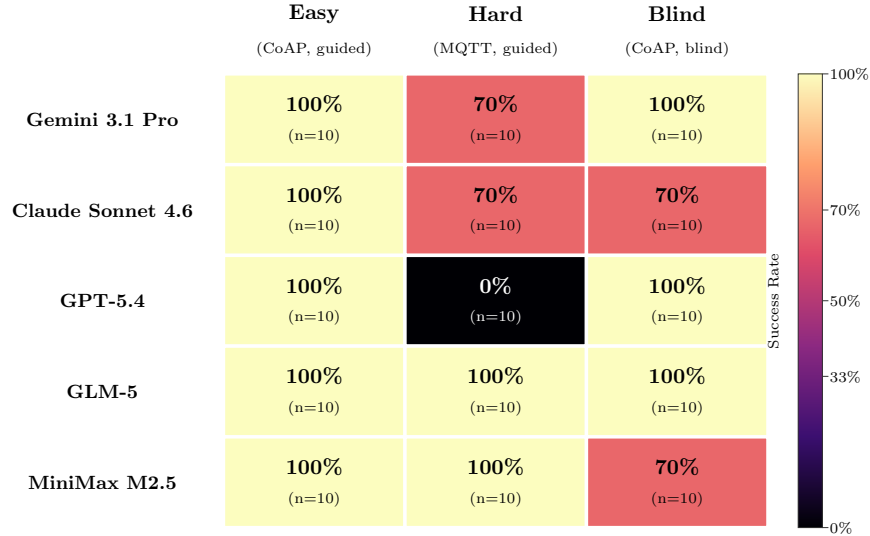


Fig. 5. Model sensitivity in mission success rate by model and condition ($n = 10$ per cell). All models achieve 100% on easy (CoAP, guided); the hard condition (MQTT, multi-step) separates models, with GPT-5.4 failing all ten runs; the blind condition (CoAP, no protocol hints) shows Claude and MiniMax M2.5 as models below 100%.

alisability concern, we conducted 150 experiment runs comparing five frontier LLMs (MiniMax M2.5, Gemini 3.1 Pro, Claude Sonnet 4.6, GPT-5.4, GLM-5) across three conditions: guided CoAP T1 (*easy*), multi-step guided MQTT T1 (*hard*), and CoAP T1 without protocol hints (*blind*). Figure 5 summarises the results, while Table 4 shows the efficiency metrics.

Easy condition (CoAP T1, guided). All five models achieved 100% success. Efficiency diverges: Gemini 3.1 Pro is fastest (7.7 turns), followed by MiniMax (12.3), GPT-5.4 (13.3), GLM-5 (15.0), and Claude Sonnet 4.6 (16.0). Gemini’s higher redundant-call rate (0.48) versus Claude (0.31) and GPT-5.4 (0.32) suggests it issued more exploratory recon calls before committing to exploitation. This easy condition establishes that all tested models possess the minimum protocol-primitive capability required to complete bare-metal MCU exploitation.

Hard condition (MQTT T1, multi-step guided). This condition is the strongest capability discriminator. GLM-5 and MiniMax M2.5 both achieved 100% success (mean 23.3 and 21.0 turns, respectively); Gemini and Claude each achieved 70% (7/10); and GPT-5.4 achieved 0% (0/10). The GPT-5.4 failures are *task-abandonment* events: all ten runs terminated within 2–3 recon calls (mission completion signalled without issuing any MQTT tool call), despite explicit multi-step guidance in the system prompt. This is a qualitatively distinct failure class from stall-and-repeat: the model exited the mission prematurely rather than

Table 4. Efficiency on successful runs (mean turns and redundant call rate). Models omitted for cells with zero successes.

Model	Condition	Success	Mean turns	Redund. rate
Gemini 3.1 Pro	easy	10/10 (100%)	7.7	0.48
Gemini 3.1 Pro	hard	7/10 (70%)	14.5	0.29
Gemini 3.1 Pro	blind	10/10 (100%)	10.3	0.43
Claude Sonnet 4.6	easy	10/10 (100%)	16.0	0.31
Claude Sonnet 4.6	hard	7/10 (70%)	31.0	0.27
Claude Sonnet 4.6	blind	7/10 (70%)	15.5	0.36
GPT-5.4	easy	10/10 (100%)	13.3	0.32
GPT-5.4	hard	0/10 (0%)	N/A	N/A
GPT-5.4	blind	10/10 (100%)	10.3	0.34
GLM-5	easy	10/10 (100%)	15.0	0.41
GLM-5	hard	10/10 (100%)	23.3	0.21
GLM-5	blind	10/10 (100%)	13.3	0.45
MiniMax M2.5	easy	10/10 (100%)	12.3	0.31
MiniMax M2.5	hard	10/10 (100%)	21.0	0.25
MiniMax M2.5	blind	7/10 (70%)	9.0	0.35

engaging the attack chain. The Gemini and Claude failures are genuine execution failures: three runs each stalled in the MQTT topic-discovery phase, identical to the overseer-OFF pattern in Section 5.3. Claude’s successful hard runs required substantially more turns (31.0 vs Gemini’s 14.5 and MiniMax’s 21.0), reflecting its more thorough state exploration before committing to each MQTT operation.

Blind condition (CoAP T1, no protocol hints). Gemini, GPT-5.4, and GLM-5 all achieved 100%; Claude and MiniMax M2.5 each achieved 70% (7/10). The Claude failures are reasoning dead-ends in the protocol-discovery phase: the model issued repeated target enumeration calls without forming a hypothesis about the CoAP option-overflow attack, eventually triggering the repetition guard. The MiniMax M2.5 blind failures are qualitatively similar: the model aborted after 4 turns without attempting exploitation. The three models achieving 100% blind success confirm that protocol-specific exploitation heuristics are not restricted to any single frontier LLM.

Summary. The easy condition was universally solved. The hard (MQTT) condition distinguishes models: GLM-5 and MiniMax M2.5 are the most capable (100%), and GPT-5.4 the least (0%, task abandonment). The blind condition showed that CoAP first-principles reasoning is accessible to most frontier models; Claude and MiniMax M2.5 showed the greatest sensitivity to the absence of protocol hints (both 70%). Across all 150 model-sensitivity runs, the overall success rate is 85.3% (128/150; 95% Wilson confidence interval: 78.8–90.1%), and aggregate success rates across the five models showed a standard deviation of 12.6 percentage points. These dispersion statistics support robustness while making model-to-model variability explicit.

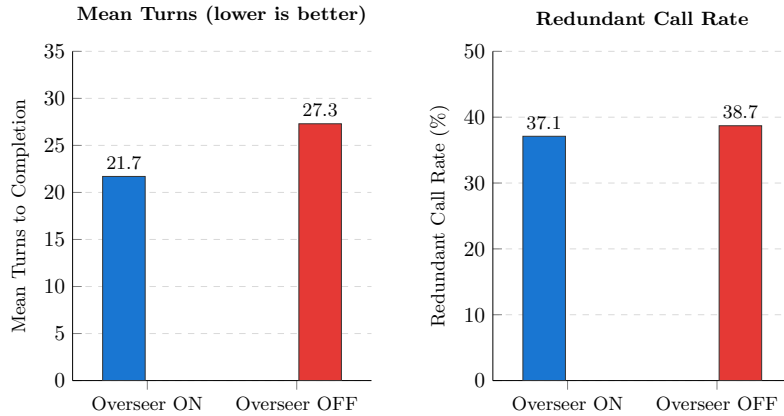


Fig. 6. Overseer ON vs. OFF comparison on T1 runs. Left: mean turns to completion, where ON is 20.5% lower, reflecting the overseer intercepting runaway loops before they accumulate. Right: redundant-call rate is nearly identical across arms (37.1% vs. 38.7%), because the LLM exhibits the same repetition tendencies with or without oversight; the overseer does not change model behaviour, it bounds its consequences. Mission success is high in both conditions (30/30 vs. 27/30); the contrast between panels is the core evidence for oversight as a structural, rather than tuning, mechanism.

Table 5. Degenerate behaviour patterns in overseer-OFF runs ($n = 30$; CoAP, Modbus, MQTT $\times 10$ replicates each).

Pattern	Runs affected	Percentage of OFF runs
Exploit without verification	30/30	100%
Repetitive same-tool loop	20/30	66.7%
Infinite recon loop (no exploit)	7/30	23.3%
No blue-team transition	3/30	10.0%
Premature phase transition	0/30	0%

5.3 Without Oversight, Degenerate Behaviour is Systematic

Figure 6 and Table 5 compare the overseer-ON and overseer-OFF conditions across 30 T1 runs each (10 CoAP + 10 Modbus + 10 MQTT [4] per condition).

With MQTT added as a third protocol, the overseer’s effect appears in both success and efficiency. Overseer-ON achieved 100% mission success (30/30; 95% Wilson confidence interval: 88.7–100.0%) versus 90.0% (27/30; 95% Wilson confidence interval: 74.4–96.5%) with oversight disabled, and completed missions in 21.7 turns on average versus 27.3 turns (20.5% fewer). The redundant-call rate is almost identical across arms (37.1% ON vs. 38.7% OFF), which is informative rather than neutral: the underlying LLM issues repeated calls at similar rates regardless of oversight, so the reduction in mission turns comes from the overseer

structurally blocking the 3rd-and-later repetitions, not from the model repeating fewer times.

Table 5 shows the stronger behavioural signal: every overseer-OFF run exhibited at least one degenerate pattern, including exploit-without-verification (100%), repetition loops (66.7%), and reconnaissance loops (23.3%). All three overseer-OFF mission failures occurred in MQTT topic discovery, where the agent stalled until timeout. The MQTT protocol is particularly sensitive to oversight absence: its multi-step attack chain (subscribe \rightarrow discover topics \rightarrow publish shutdown \rightarrow verify silence) requires correct sub-goal sequencing. Without the overseer’s phase transition enforcement, agents stalled at the subscription step. This makes MQTT the strongest demonstration of the oversight’s value in our evaluation, and motivates extending the T1 ablation to more complex protocols.

6 Further Discussions

6.1 Key Insights

The capability threshold has been crossed for bare-metal OT targets. Prior autonomous LLM pentesting work [10,14,31] operates on Linux targets where the agent can spawn shells, enumerate filesystems, and leverage familiar POSIX semantics. Bare-metal MCU targets offer none of these affordances: there is no shell to land in, no filesystem to pivot through, and success must be measured by device crash or register write. Our results show that *protocol-specific primitive tooling* is what makes this tractable: by giving the agent low-level control over CoAP option bytes and Modbus Application Protocol (MBAP) length fields, and requiring it to reason about which byte sequences trigger parser failures, APIOT bridges the semantic gap that has kept bare-metal targets out of scope for autonomous agents. An all-but-one pattern, with 100% success in 6 of 7 (protocol \times topology) conditions and a single failure condition at CoAP on three-tier topology (3/10, 30%), combined with first exploits consistently within 5 tool calls, represents a qualitative threshold: autonomous exploitation of industrial firmware is no longer a research horizon but a demonstrated capability.

For defenders, this shifts the threat model: OT security posture can no longer assume that attacker expertise is a bottleneck [20]. This conclusion holds for the emulated target class evaluated here (intentionally vulnerable Zephyr firmware on QEMU); transfer to production OT firmware with hardened parsers and real silicon is an open question addressed in the limitations.

Oversight as structural reliability mechanism, not optional tuning.

The LLM agent reliability literature [19] has identified repetition, inconsistency, and long-horizon failure as endemic to current LLMs. Proposed remedies are predominantly prompt-engineering or agent-level retries. The Claim 2 ablation showed that lightweight rule-based middleware enforcing behavioural invariants (repetition guard, crash blocker, stall detector) substantially improved reliability behaviour on the T1 baseline, and that removing it produced systematic degenerate patterns across tested conditions.

A key insight is that in 100% of overseer-OFF runs, at least one exploit call was not immediately followed by crash verification, not because the agent *could not* perform crash verification, but because without an external invariant enforcing the sequence, it would issue additional probes or transition between phases before confirming crash state. The overseer’s crash-blocker enforces this sequence structurally. The runtime governance layer reduced mission turns not by improving the LLM’s reasoning, but by intercepting a structural failure mode (issuing the same call indefinitely) that the LLM alone does not self-correct. This suggests that runtime governance is likely a horizontal engineering requirement for any safety-critical LLM agent deployment, independent of the underlying model.

6.2 Limitations

The Cortex-M3 board MAC constraint limits concurrent real QEMU MCU instances to one; we used software simulators for multi-device configurations, which were validated for crash-semantic fidelity but did not exercise QEMU-specific edge cases. Our exploit set covers overflow and credential attacks; post-exploitation persistence, privilege escalation, and lateral movement are out of scope. The testbed network was isolated; real OT network noise profiles may have differed from our Linux traffic shaping models. Primary-condition results use MiniMax M2.5, while Section 5.2 evaluates cross-model robustness and shows model-specific divergence in the MQTT hard condition. Open questions remain on: a direct primitives-vs.-named-exploits ablation, controlled quantification of cross-session memory benefits, and transfer from QEMU-emulated firmware to physical Cortex-M3 silicon.

7 Conclusion

We presented APIOT, the first end-to-end autonomous LLM agent demonstration on bare-metal MCU industrial firmware, achieving 90.0% mission success on the full discovery → exploitation → patching → verification cycle across protocols, topologies, and impairment levels, with cross-model robustness confirmed at 85.3% over five frontier LLMs. A controlled ablation showed that runtime governance is a structural reliability requirement rather than an optional tuning layer: removing it produced systematic degenerate behaviour in every run tested, while enabling it eliminated those patterns and reduced mission turns by 20.5%. The current deployment boundary—CoAP on three-tier Edge-Fog-Cloud routing—identifies multi-hop planning under stateless protocols as the next capability frontier. Both APIOT and IoT Virtual Lab will be released as open source to support reproducibility and further research.

Acknowledgments. This work was partly supported by the Engineering and Physical Sciences Research Council (EPSRC), part of the UK Research and Innovation (UKRI), under the grant numbers EP/X036871/1 and EP/X036707/1.

A System Implementation Details

A.1 APIOT Implementation Details

APIOT implements a continuous THOUGHT → ACTION → OBSERVATION loop using an OpenAI-compatible client pointed at any OpenRouter-hosted LLM. The agent maintains a growing conversation history and submits 21 tool schemas defined as JSON Schema objects to the LLM at each turn, routing the returned calls through an Action Execution Layer that maps them to Python implementations.

Two-phase tool roster. Reconnaissance tools available in both phases include `get_network_state`, `get_actionable_targets`, `stealth_check`, and `inspect_lab`. Phase 1 (Red) tools include `coap_send`, `modbus_request`, `tcp_send`, `udp_send`, `mqtt_publish`, `mqtt_subscribe`, `verify_crash`, and `verify_shell`. Phase 2 (Blue) tools include `iptables_rule`, `modbus_fc_filter`, `coap_rate_limit`, `verify_patch`, `list_patches`, and `protocol_block`. Both phases are mandatory: a mission ends only at `TASK_COMPLETE` (all targets patched and verified) or `TASK_ABORTED` (unrecoverable infrastructure failure). Linux targets are additionally reachable via `run_command` (for banner grabbing, credential testing, and post-exploitation), `remote_exec` (SSH command execution on compromised hosts), and `create_tool` (runtime Python tool generation for novel attack scenarios not covered by primitives).

Context management. Token count is estimated at 4 characters per token. When utilisation reaches 70% of the model's context window, `compaction.py` summarises older messages in-place and truncates tool results to 2 000 characters, allowing indefinitely long missions without hard token limits.

A.2 Overseer Mechanism Details

The Overseer implements seven mechanisms: (1) *Repetition guard*: blocks any (`tool`, `ip`, `args_hash`) tuple seen ≥ 3 times in a 20-call sliding window. (2) *Crash blocker*: prevents attacking a device already in "crashed" state. (3) *Patch blocker*: prevents re-attacking a vulnerability with a verified iptables rule on record. (4) *Stall detection*: injects a steering message at 8 turns without progress; forces phase transition or terminates at 15. (5) *Phase transition enforcement*: triggers the blue-team phase when all targets have been attempted; auto-signals completion when all findings are patched and verified. (6) *Strategy refresh*: every 5 turns, injects a refreshed attack plan. (7) *Overseer LLM*: an optional secondary model providing advisory support at stall events; rule-based guards are the primary enforcement mechanism.

A.3 IoT Virtual Lab Network and Device Details

IoT Virtual Lab boots real firmware images as QEMU [5] processes, connects them via Linux TAP interfaces to a shared virtual bridge (`br0`, 192.168.100.0/24),

and exposes a RESTful API for topology inspection. Three device classes are supported:

MIPS Linux routers (`qemu-system-mipsel`, Malta board): full Debian Linux with SSH and HTTP, representative of consumer router firmware.

ARM Linux gateways (`qemu-system-arm`, VersatilePB): ARM-based embedded Linux with multi-homed support, modelling edge gateway devices. A `br_internal` bridge (192.168.200.0/24) enables Purdue Model (PERA)-style segmentation.

ARM Cortex-M MCUs: Zephyr RTOS ELF images compiled with Kconfig overlays that strip IPv6 and shell to fit 64KB RAM. The testbed supports two MCU targets: Cortex-M3 (`qemu-system-arm`, `lm3s6965evb`) and Cortex-M4F (`qemu-system-arm`, `mps2-an386`). Three firmware variants provide protocol diversity: `zephyr_coap` (CoAP UDP:5683), `arm_modbus_sim` (Modbus/TCP:502), and `zephyr_echo` (echo TCP/UDP:4242). Since all exploits target the CoAP/Modbus application layer rather than CPU-specific behaviour, results generalise across both cores without separate per-core runs.

Network realism. `impair_network.sh` applies Linux `tc netem` rules to `br0` for configurable packet loss, latency, and jitter. `industrial_hmi_sim.py` generates background Modbus/CoAP traffic using a Poisson distribution, modelling SCADA polling noise.

Multi-device support and topology modes. The `lm3s6965evb` has a hardcoded MAC, limiting concurrent Cortex-M3 QEMU instances to one; lightweight Python simulators with identical crash/recovery semantics supplement QEMU for multi-MCU topologies. Four topology modes are supported: (T1) flat star, (T2) Purdue Model (PERA)-segmented (MCU behind multi-homed gateway on `br_internal`), (T3) Edge-Fog-Cloud three-tier, and a 15-node mesh. T2 and T3 require lateral movement through gateway devices to reach MCU targets.

References

1. Alghamdi, T.A., Lasebae, A., Aiash, M.: Security analysis of the Constrained Application Protocol in the Internet of Things. In: Proceedings of the 2nd International Conference on Future Generation Communication Technologies. pp. 163–168. IEEE (2013). <https://doi.org/10.1109/FGCT.2013.6767217>
2. ARM Limited: Cortex-M3 Technical Reference Manual, Revision r2p0 (2010), <https://developer.arm.com/documentation/ddi0337/latest/>
3. Arvind, S., Narayanan, A.: An overview of security in CoAP: Attack and analysis. In: Proceedings of the 2019 5th International Conference on Advanced Computing & Communication Systems. pp. 655–660. IEEE (2019). <https://doi.org/10.1109/ICACCS.2019.8728533>
4. Banks, A., Gupta, R.: MQTT version 3.1.1. OASIS Standard (2014), <https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>
5. Bellard, F.: QEMU, a fast and portable dynamic translator. In: Proceedings of the 2005 USENIX Annual Technical Conference. USENIX Association (2005), <https://www.usenix.org/conference/2005-usenix-annual-technical-conference/qemu-fast-and-portable-dynamic-translator>

6. Bhamare, D., Zolanvari, M., Erbad, A., et al.: Cybersecurity for industrial control systems: A survey. *Computers & Security* **89**, 101677:1–101677:12 (2020). <https://doi.org/10.1016/j.cose.2019.101677>
7. Chen, D.D., Woo, M., Brumley, D., Egele, M.: Towards automated dynamic analysis for Linux-based embedded firmware. In: *Proceedings of the 2016 Network and Distributed System Security Symposium*. Internet Society (2016). <https://doi.org/10.14722/ndss.2016.23415>
8. Clements, A.A., Gustafson, E., Scharnowski, T., et al.: HALucinator: Firmware re-hosting through abstraction layer emulation. In: *Proceedings of the 29th USENIX Conference on Security Symposium*. USENIX Association (2020), <https://www.usenix.org/conference/usenixsecurity20/presentation/clements>
9. Das, B.C., Amini, M.H., Wu, Y.: Security and Privacy Challenges of Large Language Models: A Survey. *ACM Computing Surveys* **57**(6) (Feb 2025). <https://doi.org/10.1145/3712001>, <https://doi.org/10.1145/3712001>
10. Deng, G., Liu, Y., Mayoral-Vilches, V., et al.: PentestGPT: Evaluating and harnessing large language models for automated penetration testing. In: *Proceedings of the 33rd USENIX Security Symposium*. pp. 847–864. USENIX Association (2024), <https://www.usenix.org/conference/usenixsecurity24/presentation/deng>
11. Fang, R., Bindu, R., Gupta, A., Kang, D.: LLM agents can autonomously exploit one-day vulnerabilities. *arXiv:2404.08144 [cs.CR]* (2024). <https://doi.org/10.48550/arXiv.2404.08144>
12. Feng, B., Mera, A., Lu, L.: P2IM: Scalable and hardware-independent firmware testing via automatic peripheral interface modeling. In: *Proceedings of the 29th USENIX Security Symposium*. pp. 1237–1254. USENIX Association (2020), <https://www.usenix.org/conference/usenixsecurity20/presentation/feng>
13. Ghazanfar, S., Hussain, F.B., ur Rehman, A., Fayyaz, U.U., Shahzad, F., Shah, G.A.: IoT-Flock: An open-source framework for IoT traffic generation. *Proceedings of the 2020 International Conference on Emerging Trends in Smart Technologies* (2020). <https://doi.org/10.1109/ICETST49965.2020.9080732>
14. Happe, A., Cito, J.: Getting pwn'd by AI: Penetration testing with large language models. In: *Proceedings of the 2023 ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. pp. 2082–2086. ACM (2023). <https://doi.org/10.1145/3611643.3613083>
15. Happe, A., Kaplan, A., Cito, J.: LLMs as hackers: Autonomous Linux privilege escalation attacks. *Empirical Software Engineering* **31**, 70:1–70:50 (2025). <https://doi.org/10.1007/s10664-025-10758-3>
16. Kim, A., Kim, D., Kim, E., Kim, S., Jang, Y., Kim, Y.: FirmAE: Towards large-scale emulation of IoT firmware for dynamic analysis. In: *Proceedings of the 2020 Annual Computer Security Applications Conference*. pp. 733–744. ACM (2020). <https://doi.org/10.1145/3427228.3427294>
17. Lai, Y., Gao, H., Liu, J.: Vulnerability mining method for the Modbus TCP using an anti-sample fuzzer. *Sensors* **20**, 2040:1–2040:20 (2020). <https://doi.org/10.3390/s20072040>
18. Li, M.Q., Fung, B.C.M.: Security concerns for large language models: A survey. *Journal of Information Security and Applications* **95**, 104284 (2025). <https://doi.org/https://doi.org/10.1016/j.jisa.2025.104284>
19. Liu, X., et al.: AgentBench: Evaluating LLMs as agents. In: *Proceedings of the 2024 International Conference on Learning Representations* (2024)
20. McLaughlin, S., Konstantinou, C., Wang, X., Davi, L., Sadeghi, A.R., Maniatakos, M., Karri, R.: The cybersecurity landscape in industrial control systems. *Proceed-*

- ings of the IEEE **104**(5), 1039–1057 (2016). <https://doi.org/10.1109/JPROC.2015.2512235>
21. Modbus Organization: Modbus application protocol specification v1.1b3. Protocol specification (2012), https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf
 22. Mohan, N., Kangasharju, J.: Edge-Fog Cloud: A distributed cloud for internet of things computations. In: Proceedings of the 2016 Congress on Cloudification of the Internet of Things. IEEE (2016). <https://doi.org/10.1109/CIOT.2016.7872914>
 23. Mou, Y., Xue, Z., Li, L., Liu, P., Zhang, S., Ye, W., Shao, J.: ToolSafe: Enhancing tool invocation safety of LLM-based agents via proactive step-level guardrail and feedback. arXiv:2601.10156 [cs.CL] (2026). <https://doi.org/10.48550/arXiv.2601.10156>
 24. Raptis, E.K., Kapoutsis, A.C., Kosmatopoulos, E.B.: Agentic LLM-based robotic systems for real-world applications: a review on their agenticness and ethics. *Frontiers in Robotics and AI* **12** (2025). <https://doi.org/10.3389/frobt.2025.1605405>
 25. Real Time Engineers Ltd: FreeRTOS - market leading RTOS for embedded systems. Website (2023), <https://www.freertos.org/>, accessed: 2025-01-15
 26. Scharnowski, T., Bars, N., Schloegel, M., Gustafson, E., Muench, M., Vigna, G., Kruegel, C., Holz, T., Abbasi, A.: Fuzzware: Using precise MMIO modeling for effective firmware fuzzing. In: Proceedings of the 31st USENIX Security Symposium. pp. 1239–1256. USENIX Association (2022), <https://www.usenix.org/conference/usenixsecurity22/presentation/scharnowski>
 27. Shelby, Z., Hartke, K., Bormann, C.: The Constrained Application Protocol (CoAP). RFC 7252, IETF (2014). <https://doi.org/10.17487/RFC7252>
 28. Stouffer, K., Pease, M., Tang, C., Zimmerman, T., Pillitteri, V., Lightman, S., Hahn, A., Saravia, S., Sherule, A., Thompson, M.: Guide to operational technology (OT) security. NIST Special Publication 800-82 Rev. 3, National Institute of Standards and Technology, Gaithersburg, MD (2023). <https://doi.org/10.6028/NIST.SP.800-82r3>
 29. Wang, H., Poskitt, C.M., Sun, J.: AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. arXiv:2503.18666 [cs.AI] (2025). <https://doi.org/10.48550/arXiv.2503.18666>, accepted to the 48th IEEE/ACM International Conference on Software Engineering (ICSE 2026)
 30. Williams, T.J.: The purdue enterprise reference architecture. *Computers in Industry* **24**(2–3), 141–158 (1994). [https://doi.org/10.1016/0166-3615\(94\)90017-5](https://doi.org/10.1016/0166-3615(94)90017-5)
 31. Xu, J., Stokes, J.W., McDonald, G., Bai, X., Marshall, D., Wang, S., Swaminathan, A., Li, Z.: AutoAttacker: A large language model guided system to implement automatic cyber-attacks. arXiv:2403.01038 [cs.CR] (2024). <https://doi.org/10.48550/arXiv.2403.01038>
 32. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: ReAct: Synergizing reasoning and acting in language models. In: Proceedings of the 2023 International Conference on Learning Representations (2023), https://openreview.net/forum?id=WE_vluYUL-X
 33. Zephyr Project: Zephyr RTOS documentation, v3.7.0. Website (2024), <https://docs.zephyrproject.org/>