Extended Abstract

Motivation Large language models (LLMs) have found many new applications in cybersecurity and can be used to greatly augment both the attack and defense of computer systems [Anthropic, 2026]. However, much existing work relies on large, state-of-the-art models that are often closed-source and expensive to run, while RL-based cybersecurity agents typically operate in abstract, game-like simulations that do not transfer to real systems. This work explores whether adversarial reinforcement learning can enhance a smaller, open LLM in an operationally realistic environment, by training an attacker and a defender agent with RL in the same containerized network against each other.

Method For both the attacker (red) and defender (blue) agents, we perform RL jointly on a Qwen3.5 language model and a CRLA (Cascaded Reinforcement Learning Agents) hierarchical policy with PPO. The CRLA policy selects a high-level network action (a target host and a tool), and the Qwen model autoregressively generates the concrete executable shell command. This factorization addresses the combinatorial complexity within each tool call's parameters, which hierarchical action-decomposition methods such as the original CRLA paper leave unaddressed.

Implementation We implement the system as a Docker containerized cyber range on a GCP virtual machine with four vulnerable target hosts (web, DNS, FTP, database) plus separate attacker and defender control containers. The targets expose realistic initial-access vectors (web flaws, anonymous FTP, weak and leaked credentials) and privilege-escalation paths (world-writable sudo scripts). In each step, the scaffold probes the range, passes the observation to the CRLA heads, injects the selected host and tool into a Qwen prompt, and executes the generated command through a bridge process that filters unsafe actions, runs the command in the appropriate container, and re-probes the range to compute rewards for PPO updates.

Results The RL-trained Qwen3.5-9B attacker is capable of learning a full privilege-escalation kill chain by dumping SSH credentials from the database, using them to log in over SSH, and abusing a world-writable sudo script to capture a root flag. It reaches credential theft in 71% of episodes and user-flag capture in 21% (versus 5% and 2% for a frozen non-RL baseline) and captured six user flags on two different hosts plus one root flag in our final run. The defender's rate of valid, cleanly-executing commands rises from 51% to 76% and it was capable of learning a layered remediation repertoire, including changing configuration files and remediating insecure permissions on files.

Discussion While reinforcement learning can somewhat improve the performance of small models on cybersecurity tasks, the reward-landscape design significantly limits high-level cyber behavior. We found that intermediate-reward bonuses repeatedly trapped the attacker in local reward hacking optima, such as repeatedly running port scans to collect a valid syntax reward or stopping once credentials or the user flag were stolen instead of pursuing the root flag. Removing process-based bonuses was necessary for the agent to reach the root flag. We also find that mean reward is not a good measure of improvement in the agents' cybersecurity capabilities: for example, the trained attacker's mean reward is below the baseline's because it forgoes safe, low-reward reconnaissance for costly exploitation that fails often but occasionally captures a flag. We therefore primarily use the metric of kill-chain progress instead of mean reward.

Conclusion Adversarial RL can push a small, open LLM toward multi-step exploitation and defense in a realistic environment, but success hinges on careful reward design in order to help the agents escape low-value local optima. Our study is limited to a single training run per configuration over relatively few PPO steps and is heavily constrained by infrastructure fragility and training time. In future work we would evaluate across multiple seeds, longer rollouts, and a wider variety of environments, toward a practical path for using smaller LLMs in operational cybersecurity.

Augmenting Cybersecurity Capabilities with Adversarial RL and Autoregressive Action Parameterization in Small LLMs

Anna Wu
Department of Computer Science
Stanford University
wuanna@stanford.edu

Ethan Ho
Department of Computer Science
Stanford University
ethanho@stanford.edu

Abstract

Large language models are increasingly capable in cybersecurity, but the strongest systems are often large, closed-source, and expensive. We study whether adversarial reinforcement learning can improve a smaller open model in an operationally realistic cyber range. We train red-team and blue-team agents with PPO in a containerized network of vulnerable hosts, where agents issue real shell commands against live services. To manage the large action space, we combine a cascaded high-level policy that selects a host and tool with an autoregressive Qwen3.5-9B policy that generates the concrete command. The trained attacker learns meaningful multi-step behavior, including dumping SSH credentials, reusing them for access, and escalating privileges to capture a root flag; it reaches credential theft in 71% of episodes and user-flag capture in 21%, compared with 5% and 2% for a frozen baseline. The defender also improves command reliability, raising its valid-command rate from 51% to 76%. However, performance is highly sensitive to reward design: intermediate bonuses can trap agents in local optima such as repeated reconnaissance or credential leakage, and mean reward can understate real capability. These results suggest that adversarial RL can improve small LLM cyber agents, but only with careful scaffolding and reward design.

1 Introduction

In recent years, large language models (LLMs) have found many new applications in cybersecurity. LLMs can be used to greatly augment both the attack [Anthropic, 2026] and defense [Castro et al., 2025] of computer systems. However, much of the existing work has focused on larger, state-of-the-art models, which are often closed-source and/or expensive to run. Furthermore, most existing RL research evaluates cybersecurity agents in simplified environments, and their conclusions cannot be easily transferred to real corporate systems [Lin et al., 2026]. This work explores how smaller LLMs can be enhanced using adversarial reinforcement learning (RL) in operationally realistic environments to improve performance in cybersecurity tasks without significantly increasing runtime costs by training both an attacker and a defender agent using RL in the same environment against each other. By training models through realistic attack–defense interactions (as opposed to more abstracted environments with simplified action spaces), we assess whether adversarial RL can offset the limitations of reduced model size, enabling compact models to approach the effectiveness of larger systems while remaining more computationally efficient.

2 Related Work

Existing studies have investigated RL for improving defensive and offensive cybersecurity agents, as well as adversarial simulations with both types of agents. Adversarial RL exposes agents to adaptive opponents and non-stationary dynamics and can produce more resilient defensive strategies and more realistic attack behaviors compared to static training regimes [Nguyen and Reddi, 2020, Landolt et al., 2025].

While recent work has demonstrated that frontier LLMs can perform impressively in realistic cybersecurity settings [Lin et al., 2026], these models are extremely computationally expensive; conversely, studies that apply RL to cybersecurity agents improve computational costs but sacrifice realism. RL studies overwhelmingly use simplified game-like simulation environments such as the CyberBattleSim [Microsoft Defender Research Team, 2021] and CybORG [Standen et al., 2021], where agents' actions are heavily abstracted—for example, offensive actions may be represented by choosing a MITRE ATT&CK technique [Dutta et al., 2023] or simply selecting an IP to target [Oh et al., 2023, Shahid et al., 2025] with no introspection into the mechanics of the actual exploit. In most cases, agents never directly enter commands or interact with a filesystem, and rewards are given for reaching a predefined success node on a graph [Oh et al., 2023, Zhang et al., 2026], not a concrete proof of compromise such as access to a protected file. In experiments with adversarial competition, the environment is almost always a game where the attacker and defender alternate in rounds [Shahid et al., 2025, Zhang et al., 2026], which does not realistically represent real-world engagements where actions happen concurrently. These abstractions severely limit the applicability of previous RL cybersecurity agents to real environments.

A chief challenge with applying RL to cybersecurity is the combinatorial explosion of the realistic action space. Each tool call has numerous syntactically specific parameters, and networks consist of numerous hosts to enumerate, defend or exploit. Addressing the network complexity problem, Tran et al. [2022] propose the Cascaded Reinforcement Learning Agent (CRLA) architecture, which decomposes the action space using a binary segment tree and a hierarchy of DQN agents. However, CRLA does not address the dimension of combinatorial complexity within each tool call's parameters. This is the problem that we address in this study.

3 Method

We trained both an attacker (red team) and a defender (blue team) agent using RL to perform cybersecurity tasks, with the attacker trying to penetrate into a computer network and the defender model fixing vulnerabilities in the same network. In training both models together, the defender model is penalized when the attacker model makes progress, and the attacker model has a harder time penetrating systems when the defender model secures systems. For both attacker and defender agents, we performed RL jointly on a Qwen3.5 model and CRLA model. The CRLA model provides general guiding actions, and the Qwen model generates the actual executable commands. Both CRLA and Qwen are trained jointly with PPO.

3.1 Agent Scaffolds

Figure 1 shows the full agent scaffold used to connect the learned policies to the cyber range. During each PPO iteration, both the red and blue agents follow the same high-level loop. First, the scaffold builds an observation by running fixed status and inspection commands against the range machines. These commands collect information about reachable hosts, running services, service health, previous command outputs, and other state needed by the agent. This observation is passed to the CRLA layer, which makes a structured high-level decision about the next network action. In our implementation, this consists of a target-host selection head followed by a tool-selection head, producing a pair such as $(\text{host}, \text{tool})$.

The selected CRLA action is then inserted into the prompt layer for Qwen. The prompt is split into a system portion and an agent-specific portion. The system portion specifies formatting requirements and execution constraints, while the agent-specific portion describes whether the model is acting as the red attacker or blue defender, the agent's objective, and relevant history from the current episode. Conditioned on this prompt, Qwen generates a concrete bash command to execute. For the attacker (red) agent, the command is executed from a Kali Linux container. For the defender (blue) agent, the

command is executed from a management container that can administer the machines in the cyber range.

Commands are sent through an HTTP tunnel to the GCP VM hosting the experiment. A bridge process receives each command, filters destructive or out-of-scope actions, executes the remaining command in the appropriate container, and then probes the cyber range again to measure the resulting state. The scaffold then calculates the environmental reward, the potential-based shaping reward, and the intrinsic command-quality reward. These rewards are assembled into a final scalar reward that is used to update both the CRLA heads and the Qwen command policy.

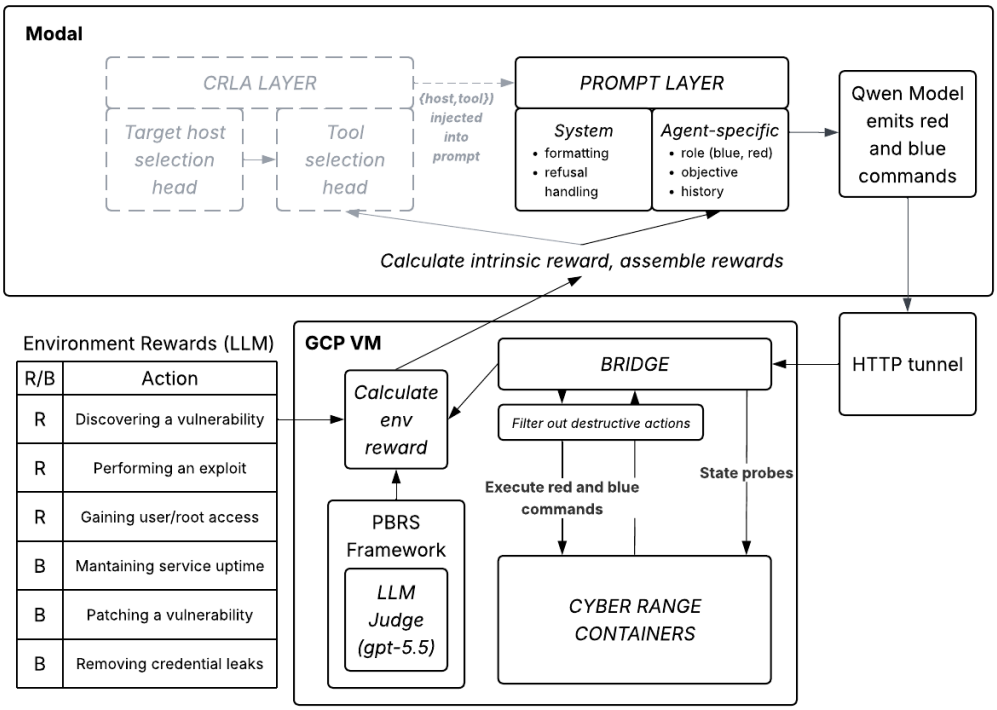

Figure 1: RL infrastructure architecture.

3.2 Algorithms and Rewards

Our system uses a hierarchical policy scaffold for both the attacker and defender agents. At each PPO iteration, the agent first constructs an observation o by running fixed status and inspection commands over the cyber range. These probes collect information such as reachable hosts, running services, open ports, process state, service health, and prior command history. The observation is passed into a cascaded CRLA policy layer, which selects a structured network action a_{net}. This action contains both a target host decision and a tool decision, such as choosing which machine to inspect, attack, patch, restart, or defend.

The selected network action is then injected into a prompt for Qwen. The prompt contains a system layer, which enforces formatting and refusal-handling behavior, and an agent-specific layer, which specifies whether the model is acting as the red attacker or blue defender. Conditioned on the observation and CRLA-selected network action, Qwen autoregressively emits a concrete shell command $a_{1:T}$ to execute in the range:

$$\log \pi(a \mid o) = \log \pi_{\text{CRLA}}(a_{\text{net}} \mid o) + \sum_{t=1}^{T} \log \pi_{\text{AR}}(a_t \mid o, , a_{\text{net}}, , a_{1:t-1}).$$

This factorization separates high-level cyber strategy from low-level command generation. The CRLA heads learn which host and tool class are useful in the current state, while the language model

learns to instantiate that decision as a valid command. Commands are executed either inside the red Kali container, for attacker actions, or inside the blue management container, which can administer the cyber range machines. Before execution, the bridge filters destructive or out-of-scope commands.

Both the CRLA layer and Qwen command policy are trained on-policy. After each command, the range is inspected again and rewards are assigned to both agents. We experiment with both state-only rewards and state-action rewards. State-only rewards depend on the resulting range state, while state-action rewards also account for the command that produced the transition. The base environmental reward is either assigned from a static rule list or by an LLM judge. For the attacker, positive rewards are given for discovering vulnerabilities, demonstrating exploitation evidence, and gaining user or root access. For the defender, positive rewards are given for maintaining service uptime, patching vulnerabilities, and removing credential leaks. The defender is penalized when the attacker obtains access, while the attacker is penalized for failed or ineffective attacks.

To reduce sparsity in these rewards, we add potential-based reward shaping (PBRS). We estimate a state potential $\Phi(s)$ using an LLM judge that only observes the range state, not the command itself. The shaped reward is

$$r_{\text{PBRS}}(s_t, a_t, s_{t+1}) = r_{\text{LLM/env}}(s_t, a_t, s_{t+1}) + \beta\left(\gamma\Phi(s_{t+1}) - \Phi(s_t)\right).$$

The potential term provides dense progress feedback: states that appear closer to the agent's objective receive higher potential, while regressions receive lower potential. Since the potential judge only uses observations, the shaping term is intended to reward progress in the cyber range rather than surface-level properties of a command. Under the standard PBRS condition, where Φ is a potential function over Markov states and the same discount factor γ is used, this shaping transformation preserves the optimal policy for the original environmental reward Ng et al. [1999].

Finally, we include an intrinsic command-quality reward:

$$r_{\text{total}} = r_{\text{PBRS}}(s_t, a_t, s_{t+1}) + \lambda r_{\text{int}}(a_{\text{net}}, a_{1:T}).$$

This term penalizes malformed shell syntax, commands that time out, commands that fail to execute, and commands that are inconsistent with the selected target or tool. It rewards commands that are valid, executable, and well aligned with the CRLA action. Unlike PBRS, this intrinsic term is not intended to preserve the original cyber objective; instead, it acts as an auxiliary training signal that improves the reliability of Qwen's command generation.

The final scalar reward is used to update both the high-level CRLA heads and the autoregressive Qwen policy. Table 1 summarizes the concrete reward landscape used for the attacker and defender.

4 Experimental Setup

We built a "cyber range" consisting of four machines with real vulnerable network services running on them, simulating the computer systems for a small company. The attacker agent is connected over the network to these machines, and the defender agent has access to the management interface for each machine, allowing command execution on any of the four machines. Figure 2 shows the machines in the cyber range, as well as their relation to the machines that the attacker and defender agents are given access to.

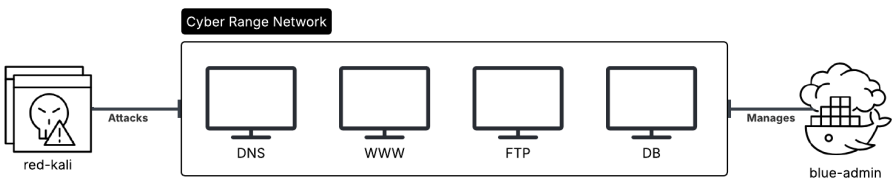

Figure 2: Cyber range architecture.

Each machine has vulnerabilities on it that could lead to user or root-level compromise. These include both initial access vectors that allow for an attacker to gain remote code execution (for example,

Component	Condition	Value
Attacker kill-chain potential Φ_{red} (defender receives $-\Phi$)		
Rung 0	Reconnaissance only	0.0
Rung 1	SSH credentials leaked into output	0.5
Rung 2	Code execution on a target (`uid=`)	1.5
Rung 3	User flag captured (`/opt/flag.txt`)	3.5
Rung 4	Root flag captured (`/root/flag.txt`, via privesc)	8.0
Defender securing (subtracted from attacker advantage)		
Vulnerability closed	per class (LFI, injection, anon FTP, SQLi, webshell, creds leak, weak DB auth, privesc)	$+0.4$ to $+0.5$
Service-down guardrail	per required service taken offline	$+0.4$ to attacker
Intrinsic command-quality r_{int} (per-agent, weight $\lambda = 0.1$)		
Valid command	parses successfully	$+0.15$
Clean execution	exit code 0	$+0.10$
Repeated reconnaissance	recon tool reused after ≥ 2 recent scans	-0.80
Exact command repeat	command seen earlier in episode	-0.50
Invalid command	fails to parse	-0.75
Timeout	command exceeds time limit	$-1.5\times$
Other		
Invalid-command forfeit	replaces env+intrinsic for invalid moves	-1.0
PBRS shaping	LLM-judge state potential	$\beta(\gamma\Phi(s') - \Phi(s))$

Table 1: Reward landscape. The attacker's environmental reward is telescoped, making achieving the next successive rung in the kill chain the only way to gain reward instead of simply repeating a single exploit. The defender receives the negation of the attacker's reward, plus credit for closing vulnerabilities. An auxiliary intrinsic term helps to shape the generated command quality, and is deliberately small so that a long episode of valid-but-inert commands cannot out-earn an actual capture. A flag is credited only when the true flag secret appears in the attacker's output, which prevents reward hacking by echoing the flag marker.

Host	Vector (service / file)	Vulnerability	Access gained
WWW	`download.php`	Path traversal / arbitrary file read	Leak `creds.php`, readable files
	`ping.php`	Command injection	RCE as `www-data`
	`search.php`	SQL injection (into Postgres)	Read DB, dump `ssh_accounts`
	`upload.php`	Unrestricted file upload	Webshell, RCE as `www-data`
	`includes/creds.php`	Credential disclosure	SSH logins for all hosts
	SSH (`webdev`)	Leaked credentials	User shell
	`www-backup`	World-writable sudo script	**Root (privesc)**
DNS	SSH (`dnsops`)	Leaked credentials	User shell
	`dns-zone-backup`	World-writable sudo script	**Root (privesc)**
FTP	vsftpd (`anon_root=/`)	Anonymous FTP, filesystem root exposure	Read arbitrary files, creds dump
	SSH (`ftpuser`)	Leaked credentials	User shell
	`ftp-archive`	World-writable sudo script	**Root (privesc)**
DB	PostgreSQL (`root:root`)	Weak superuser credentials	DB superuser, `COPY` RCE
	`ssh_accounts` table	Credential disclosure	SSH logins for all hosts
	SSH (`dbreport`)	Leaked credentials	User shell
	`db-maintenance`	World-writable sudo script	**Root (privesc)**

Table 2: Vulnerabilities by host. Each host exposes one or more *initial access vectors* (web flaws, anonymous FTP, weak/leaked credentials) and a *privilege-escalation path* (a world-writable script runnable as root via `sudo`). The user flag `/opt/flag.txt` is readable after an SSH login as the host's user. The root flag `/root/flag.txt` requires the privilege-escalation step. The wide, overlapping attack surface reduces reward sparsity, as patching any single bug does not close the path to compromise.

vulnerabilities in the web server, FTP server, or database, or leaked credentials that allow for SSH login) and for privilege escalation (world-writable scripts that can be run with sudo, SUID binaries). The environment as a whole is designed so that there are many different attack vectors and a wide attack surface so that patching any one bug does not prevent the attacker from making progress. This is done to reduce the sparsity of the rewards, as there will be many opportunities to make progress in the environment from an attacker perspective, and many opportunities to secure the machines.

5 Results

We focus our evaluation on two configurations: a non-RL **baseline** (frozen Qwen3.5-9B run in the same environment with no gradient updates) and **Qwen3.5-9B** trained with adversarial RL. We also briefly experimented with a smaller Qwen3.5-2B model, but its command generation was so unreliable that RL drove it to game the reward signal with inert connection tests rather than real cyber actions; we defer that analysis to Appendix C. We measure attacker progress using a *kill-chain rung*, the highest level of compromise the attacker demonstrates within an episode (Table 1): rung 0 reconnaissance only, rung 1 credentials leaked, rung 2 code execution on a target, rung 3 user-flag capture, and rung 4 root-flag capture via privilege escalation.

5.1 Quantitative Evaluation

Main results. Table 3 reports a capability profile for the baseline and the RL-trained 9B agent with the fraction of episodes that reach each rung. The RL attacker reaches credential theft in 71% of episodes (vs. 5% for the baseline) and user-flag capture in 21% (vs. 2%), and is the only configuration ever to reach a root flag. All numbers are from a single training run per configuration.

Configuration	Episodes reaching rung (%)				Blue valid
	$\geq$1 creds	$\geq$2 exec	$\geq$3 user	$\geq$4 root	cmd (%)
Baseline (9B, frozen)	5	5	2	0	51
Qwen3.5-9B + RL (ours)	**71**	**21**	**21**	**3**	**76**

Table 3: Capability profile: the fraction of episodes in which the attacker reaches each kill-chain rung (credentials leaked, code execution, user-flag capture, root-flag capture), plus the defender's valid-command rate. The non-RL baseline is a frozen model and serves as the no-training control. The RL attacker reaches credential theft in most episodes, user-flag capture in roughly one in five, and is the only configuration to ever reach a root flag; the defender's command reliability rises from 51% to 76%. Totals over the run: the RL agent captured six user flags and one root flag (first root at update 4) versus one user flag and no root flags for the baseline.

Per-host compromise. Table 4 breaks down attacker captures by target machine for the 9B RL agent. Captures are not limited to a single box: the attacker obtained user flags on multiple hosts (using the distinct leaked SSH credentials for each) and escalated to root on the web host by abusing its world-writable sudo script.

Host	Initial access used	User flags	Root flags
WWW	SSH (`webdev`) via leaked creds	3	1
DNS	SSH (`dnsops`) via leaked creds	3	0
FTP	SSH (`ftpuser`) via leaked creds	0	0
DB	SSH (`dbreport`) via leaked creds	0	0

Table 4: Attacker captures by host for Qwen3.5-9B + RL (counts over the training run). Compromise is not limited to one box: the attacker captured user flags on two hosts using each host's distinct leaked SSH credential, and escalated to root on WWW by appending a `cat /root/flag.txt` payload to the world-writable `/usr/local/bin/www-backup` sudo script and invoking it.

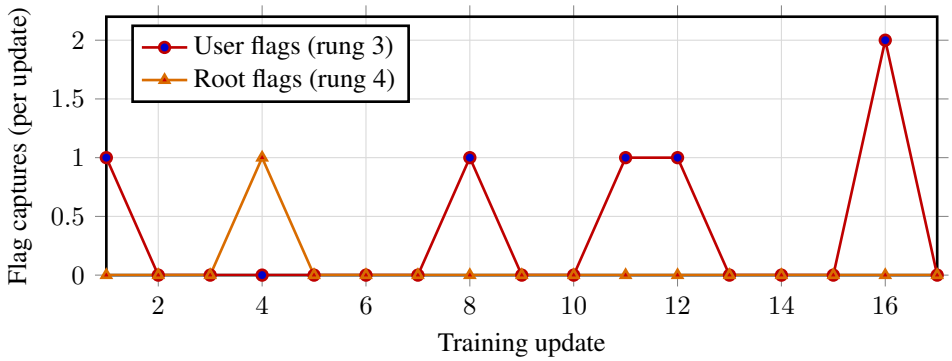

Figure 3: Flag captures per training update for Qwen3.5-9B + RL (6 user flags, 1 root flag over the run; the baseline captured a single user flag and no root flags across a longer run). User-flag captures cluster in the later half of training. Importantly, we report captures rather than mean reward, which is dominated by the zero-sum/forfeit dynamic and does not reflect capability (Appendix B).

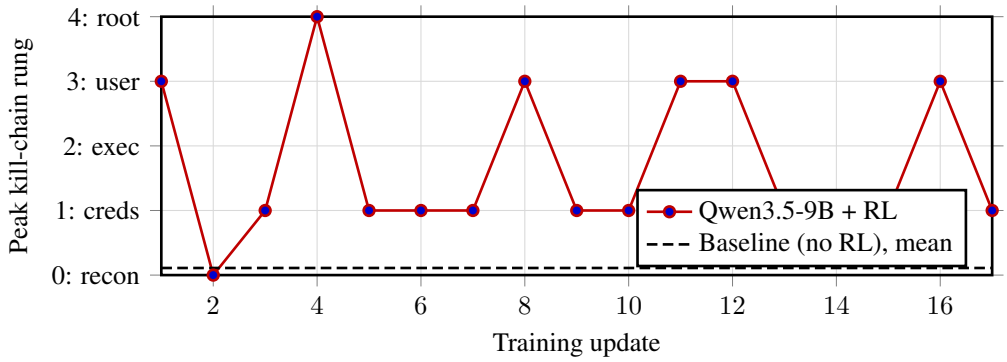

Figure 4: Peak kill-chain rung reached per training update for Qwen3.5-9B + RL, with the baseline's mean (dotted). This is the clearest learning-progression view: the RL agent repeatedly reaches credential theft (rung 1), climbs to user-flag capture (rung 3) in several episodes, and reaches a root-flag capture (rung 4) at update 4, whereas the baseline almost never leaves reconnaissance (mean rung 0.11).

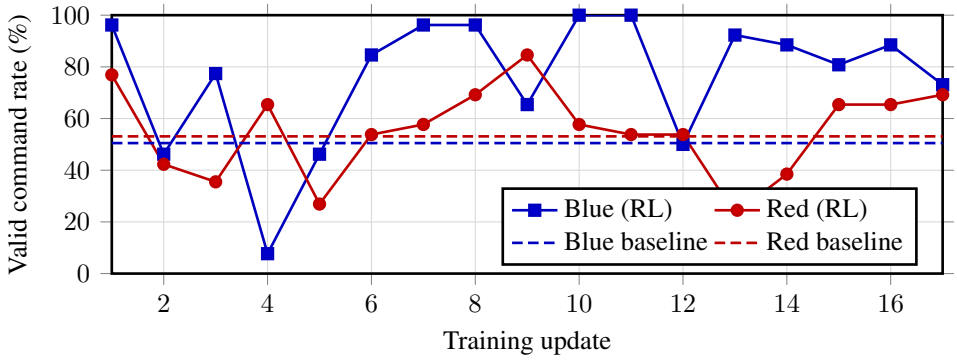

Figure 5: Fraction of generated commands that parse and execute cleanly, per training update, with baseline means (dotted). The defender's command reliability rises well above the baseline (mean 76% vs. 51%), reflecting that RL teaches it to issue valid, executable remediation commands. The attacker's validity stays near the baseline (55% vs. 53%): rather than producing *safer* commands, the RL attacker spends its budget attempting harder, higher-value exploitation steps (SSH logins, privilege escalation) that fail more often but, when they succeed, advance the kill chain.

Learning progression. Figures 3–5 trace how this behavior emerges over training. The peak-rung curve (Figure 4) shows the attacker moving off pure reconnaissance into reliable credential theft and, increasingly, user-flag capture; the flag curve (Figure 3) shows captures concentrating in the later half of training; and the validity curve (Figure 5) shows the defender becoming markedly more reliable at issuing executable commands while the attacker trades safety for higher-value attempts.

5.2 Qualitative Analysis

The attacker learns a full kill chain. The 9B RL attacker progresses through clearly identifiable stages over training. Early updates are dominated by reconnaissance (port scans with `nc`); once the anti-recon penalty makes pure scanning unprofitable, the attacker shifts to credential discovery, settling on the database channel (`psql ... SELECT * FROM ssh_accounts`) as a reliable source of SSH logins. It then learns to reuse the leaked credentials to SSH into a target and read the user flag, and in its strongest episode chains this into a privilege escalation by appending a `cat /root/flag.txt` payload to a world-writable sudo script and executing it as root. A representative end-to-end root-capture trajectory is shown in Appendix A.1.

Reward shaping and local optima. The attacker's behavior is highly sensitive to intermediate reward design. Initially, we provided a larger per-step command-validity bonus which caused the attacker to collapse into reward hacking by cycling through port scans that accrued small positive reward over a long episode; we addressed this by reducing the bonus and penalizing repeated reconnaissance. Likewise, a one-time credential-leak bonus created another local optimum where the attacker leaked credentials and stopped, because the leak alone was a satisfying payoff. Removing that bonus so that leaked credentials are valuable only as a means to SSH and escalate was necessary for the attacker to advance to capturing the root flag. These experiments show that, for small models, reward-landscape design is extremely influential in the model's cybersecurity capabilities. They also explain why mean reward is a poor capability metric, as the trained attacker's mean reward is actually lower than the baseline's (Appendix B) because it forgoes safe, near-zero-reward reconnaissance in favor of costly exploitation attempts that frequently fail but occasionally capture a flag.

The defender learns reliable, layered defense. The defender's most measurable improvement is command reliability: its valid, cleanly-executing command rate rises from 51% at baseline to 76% (Figure 5). Qualitatively, over the course of training it employs the full defensive repertoire—removing vulnerable PHP scripts, disabling anonymous FTP, changing the database password and dropping the leaked credential table, applying configuration-level PHP hardening (`disable_functions`, `open_basedir`) and `.htaccess` WAF rules, and locking the world-writable privilege-escalation scripts. Within any single episode, however, it tends to repeat its single most reliable fix rather than diversify, so its per-episode breadth does not exceed the (less reliable, more scattershot) baseline—an asymmetry consistent with the attacker facing a longer, more brittle multi-step objective than the defender. A concurrent attack–defense episode in which both agents act meaningfully is shown in Appendix A.2.

6 Discussion

6.1 Findings

Our experiments suggest that adversarial RL for small LLM cyber agents can show meaningful improvement over non-RL baselines, but is bottlenecked by action validity and reward design. The smallest models (e.g., 2B) struggle most with even generating valid commands, and RL can push them to the simplest possible commands that earn small rewards, "giving up" on higher-value cyber actions that the model cannot learn. With a larger instruction-tuned model and a more stable scaffold, however, RL was able to produce more operationally meaningful behavior.

Training Stability. We found that the main challenge was not only cyber task completion, but maintaining a coherent command-generating policy during RL. Sparse or mostly negative rewards caused the language model's output distribution to rapidly degrade, as agents began producing malformed shell syntax, unterminated quotes, markdown code block openings, and commands unrelated to the environment. Adding intrinsic rewards for commands that executed successfully helped prevent this collapse and produced more stable early training. However, these rewards had to

remain small, since overly rewarding valid execution encouraged safe but inert commands rather than cyber progress.

Reward Shaping. Several reward designs that we experimented with produced unintended behavior. Initially, attacker agents received a positive reward when the vulnerability probes showed that exploitable conditions still existed. This rewarded inaction on the part of the attacker, as commands that did not change the environment but only examined the state of network services could still receive positive feedback. We lessened this reward and instead tied attacker rewards to exploitation evidence and access progress. Conversely, overly stingy rewards with almost no positive signal led to rapid degeneration, showing that small LLM policies require some dense syntactic and execution-level feedback before meaningful cyber rewards can be learned. Overall, reward shaping controlled not only training stability, but also which level of cyber behavior the agent converged to; small intermediate bonuses could create local optima around reconnaissance or credential leakage rather than completion of the full task.

Scaffolding Effects. Prompt and environment scaffolding were critical. Base, non-instruction-tuned models were unable to reliably follow the task format, so we moved to instruction-tuned Qwen models. Explicitly including in the Qwen models' prompts that it was not already on a target host, and that it needed to use SSH, Telnet, network tools, or `docker exec` depending on role, reduced invalid local actions. We also added command timeouts, command filtering, range probes, and pre-provisioned common tools to improve training and evaluation performance.

Model Scale. The smaller Qwen models struggled to produce coherent operational commands. Moving from sub-billion-parameter models to Qwen3.5-2B improved basic command coherence, but did not fully solve long-horizon task execution. The strongest behaviors for the smaller models remained shallow reconnaissance actions such as host discovery, port checks, and simple service inspection rather than complete exploitation or remediation chains. Moving to Qwen3.5-9B made the task more feasible, but did not remove the need for careful reward and scaffold design.

CRLA Behavior. Our CRLA experiments show that decomposing the action space into host and tool choices can be helpful for controlling a large action space, but can also be detrimental. In practice, it often switched targets or tools from step to step. Cybersecurity tasks often require multi-step commitment: an agent must pursue the same host, credential, service, or vulnerability chain across several actions. A hierarchy reselecting independent host-tool pairs at every step can work against this requirement. Limiting the possible tools by role helped reduce obviously inappropriate tool choices, especially for the defender, but the CRLA policy switching hosts frequently prevented multi-step processes from playing out.

6.2 Limitations and Future Work

In this study, we were not able to run very long rollouts due to the fragility of the infrastructure and ran relatively few PPO steps. Furthermore, all the headline results come from a single training run per configuration, so they should just be interpreted as evidence that adversarial RL can improve small LLM cyber agents in this environment, not as a precise estimate of the method's average performance. Even though our cyber range training environment is somewhat realistic for infrastructure used by a small company, having a larger variety of environments for training would be helpful to improve generalization of results.

Also, our reward structure underwent a large amount of tuning to improve performance, which may not be possible in a real-world deployment setting. In future work, we would like to train our agents in more environments with more generalized rewards, run longer rollouts and more PPO iterations, and evaluate across multiple random seeds and model sizes. We would also like to improve the hierarchical policy so that it can commit to a host, credential, or vulnerability chain over multiple steps instead of reselecting independent host-tool pairs at every action. This would allow for more generalizable results than our smaller-scale study, and would allow for evaluation of how useful adversarial RL can be for real-world penetration testing and active cyber defense. Through these extensions, we hope that this technique could contribute to a practical way to use smaller LLMs for operational cybersecurity.

7 Conclusion

In this work, we explored whether adversarial reinforcement learning can improve the cybersecurity capabilities of smaller open language models in an operationally realistic environment. We trained red-team and blue-team agents in a containerized cyber range where agents issued real shell commands against live services, using a hierarchical CRLA policy for host and tool selection together with an autoregressive Qwen policy for command generation. Our results show that adversarial RL can move a small model beyond shallow reconnaissance toward meaningful multi-step cyber behavior, including credential theft, SSH access, user-flag capture, and limited privilege escalation. The defender also became more reliable at producing executable remediation commands.

At the same time, our experiments show that this improvement depends heavily on reward design and scaffolding. Small changes in intermediate rewards could trap agents in local optima such as repeated reconnaissance or credential dumping, while overly sparse rewards caused command generation to degrade. These findings suggest that adversarial RL is a promising but fragile approach for improving small LLM cyber agents. Future work should evaluate the method across more environments, longer training runs, and multiple seeds, with more general reward functions and hierarchical policies that better support long-horizon commitment. Overall, our study provides evidence that smaller open models can be pushed toward operational cybersecurity tasks, but only when paired with careful infrastructure, reward design, and evaluation.

8 Team Contributions

- **Anna Wu:** Designed evaluation infrastructure and agent structure. Created reward structure for both red/blue agents and prompt structure for Qwen command generation.
- **Ethan Ho:** Created cyber range infrastructure and vulnerabilities. Contributed to evaluation infrastructure and designed LLM-as-judge reward scoring.

Changes from Proposal We ended up using a larger Qwen model (Qwen3.5-9B instead of Qwen3.5-0.8B), and softened our rewards to be not exactly zero sum. This is because we found that the LLMs struggled to create syntactically valid commands, and needed to both increase model size and add rewards for successful commands to encourage meaningful behavior.

AI Tools Disclosure Claude Code (Claude Sonnet 4.6) and Codex (GPT-5.5) were used for infrastructure scaffolding (Modal GPU integration, HTTP bridge server, Docker environment management, debugging). The PPO algorithm, CRLA implementation, reward functions, and research direction were developed independently. We also used the ChatGPT web interface for writing assistance, mainly for rewording and rephrasing some of our ideas that we previously kept in a looser notes/outline format.

References

Anthropic. Claude mythos preview system card. Technical report, Anthropic, April 2026. URL https://www-cdn.anthropic.com/08ab9158070959f88f296514c21b7facce6f52bc.pdf. Accessed: 2026-04-29.

Sebastián R. Castro, Roberto Campbell, Nancy Lau, Octavio Villalobos, Jiaqi Duan, and Alvaro A. Cardenas. Large language models are autonomous cyber defenders. In *2025 IEEE Conference on Artificial Intelligence (CAI)*, page 1125–1132. IEEE, May 2025. doi: 10.1109/cai64502.2025.00195. URL http://dx.doi.org/10.1109/CAI64502.2025.00195.

Ashutosh Dutta, Samrat Chatterjee, Arnab Bhattacharya, and Mahantesh Halappanavar. Deep reinforcement learning for cyber system defense under dynamic adversarial uncertainties, 2023. URL https://arxiv.org/abs/2302.01595.

Christoph R. Landolt, Christoph Würsch, Roland Meier, Alain Mermoud, and Julian Jang-Jaccard. Multi-agent reinforcement learning in cybersecurity: From fundamentals to applications, 2025. URL https://arxiv.org/abs/2505.19837.

Justin W. Lin, Eliot Krzysztof Jones, Donovan Julian Jasper, Ethan Jun shen Ho, Anna Wu, Arnold Tianyi Yang, Neil Perry, Andy Zou, Matt Fredrikson, J. Zico Kolter, Percy Liang, Dan Boneh, and Daniel E. Ho. Comparing ai agents to cybersecurity professionals in real-world penetration testing, 2026. URL https://arxiv.org/abs/2512.09882.

Microsoft Defender Research Team. CyberBattleSim. https://github.com/microsoft/CyberBattleSim, 2021. Accessed: 2025.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann.

Thanh Thi Nguyen and Vijay Janapa Reddi. Deep reinforcement learning for cyber security: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2020. Includes discussion of multi-agent RL in cybersecurity contexts.

Sang Ho Oh, Min Ki Jeong, Hyung Chan Kim, and Jongyoul Park. Applying reinforcement learning for enhanced cybersecurity against adversarial simulation. *Sensors*, 23(6):3000, 2023. doi: 10.3390/s23063000.

Abrar Shahid, Ibteeker Mahir Ishum, AKM Tahmidul Haque, M. Sohel Rahman, and A. B. M. Alim Al Islam. Adversarial reinforcement learning for offensive and defensive agents in a simulated zero-sum network environment, 2025.

Maxwell Standen, Martin Lucas, David Bowman, Toby J. Richer, Junae Kim, and Damian Marriott. Cyborg: A gym for the development of autonomous cyber agents, 2021. URL https://arxiv.org/abs/2108.09118.

Khuong Tran, Maxwell Standen, Junae Kim, David Bowman, Toby Richer, Ashlesha Akella, and Chin-Teng Lin. Cascaded reinforcement learning agents for large action spaces in autonomous penetration testing. *Applied Sciences*, 12(21):11265, 2022. doi: 10.3390/app122111265.

Yiyao Zhang, Diksha Goel, and Hussain Ahmad. Explainable autonomous cyber defense using adversarial multi-agent reinforcement learning, 2026.

A Sample Trajectories

A.1 Example of a root flag capture

The episode below (training update 4) is the one in which the Qwen3.5-9B attacker captured a root flag. We show the attacker's commands at each joint step, annotated with what it was attempting and the per-step telescoped environmental reward r. In summary, the attacker dumps SSH credentials from the database, reuses those credentials to read user flags over SSH on two hosts, and finally escalates to root on the web host by abusing a world-writable sudo script.

```
STEP 1  [recon, r=-1.48]
  nc -vz db.company.com 5432
  -> Probes the Postgres port to confirm the database is reachable. A pure scan,
     penalized by the anti-reconnaissance shaping term.

STEP 2  [RUNG 1: credentials leaked, r=+0.51]
  psql 'postgresql://root:root@db.company.com:5432/company' -c 'SELECT * FROM ssh_accounts;'
  -> Uses the weak root:root database login to dump the ssh_accounts table,
     leaking every host's SSH username and password into its output.

STEP 3  [r=+0.01]
  psql '...company' -c '\d ssh_accounts'
  -> Inspects the table schema, orienting itself to the leaked data.

STEP 4  [r=+0.01]
  psql '...company' -c 'COPY ssh_accounts TO STDOUT WITH CSV HEADER' > ~/credentials.csv
  -> Saves the dumped credentials to a local file for later use.

STEP 5  [r=+0.01]
  psql '...company' -f ~/credentials.csv
  -> Misfire: tries to execute the saved CSV as a SQL script; no effect.

STEP 6  [RUNG 3: user flag on DNS, r=+2.98]
  sshpass -p resolver2026 ssh -o StrictHostKeyChecking=no dnsops@dns.company.com 'cat /opt/flag.txt'
  -> Reuses the leaked dnsops credential to SSH into the DNS host and read its
     user flag -- the first true compromise.

STEP 7  [r=-0.02]
  psql '...company' -c "\copy sshaccounts(to stdout)" || true
  -> Attempts an alternative credential re-dump. It uses a malformed table name, so this has no effect.

STEP 8  [r=-0.02]
  psql postgresql:///postgres -qtc "...SELECT host,username,password..." && sshpass -p $(...) ssh webdev@..
  -> Tries to combine a credential dump and an SSH login into one command;
     malformed, fails (rc=2).

STEP 9  [r=-0.02]
  sshpass -p ftpguest2026 ssh -o StrictHostKeyChecking=no ftpuser@ftp.company.com 'cat /opt/flag.txt'
  -> Reuses the leaked ftpuser credential to read the FTP host's user flag.

STEP 10 [r=-0.02]
  # (emits reasoning instead of a command; no-op)

STEP 11 [RUNG 4: ROOT FLAG on WWW, r=+4.43]
  sshpass -p webdev2026 ssh -o StrictHostKeyChecking=no webdev@www.company.com \
     'echo "cat /root/flag.txt" >> /usr/local/bin/www-backup && sudo /usr/local/bin/www-backup'
  -> The privilege escalation: SSHes into the web host as webdev, appends a
     flag-read payload to the world-writable www-backup sudo script, and runs it
     as root -- reading the protected root flag.
```

A.2 A full sample trajectory

The episode below illustrates an example of a full trajectory. In this sample, notable events include the attacker capturing a user flag and the defender removing various scripts and hardening actions. Each step shows the concurrent attacker (RED) and defender (BLUE) commands.

```
STEP 1
  RED : nc -zv -w 2 db.company.com 5432             # recon: probe Postgres
  BLUE: docker exec cyberrl-blue-admin ps aux --forest   # inspect processes

STEP 2  [RED RUNG 1: credentials leaked]
  RED : psql root:root@db...company -c 'SELECT * FROM ssh_accounts;'   # dump SSH creds
  BLUE: docker exec company-www rm -f /var/www/html/get.php            # remove LFI script

STEP 3
  RED : nc -zv -w 2 www.company.com 22              # probe WWW SSH
  BLUE: docker exec company-www rm -f /var/www/html/downloads.php      # remove LFI script

STEP 4
  RED : nc -zv -w 2 www.company.com 80              # probe WWW HTTP
  BLUE: docker exec company-www rm -f /var/www/html/files.php          # remove LFI script

STEP 5  [RED RUNG 3: user flag on WWW]
  RED : sshpass -p webdev2026 ssh webdev@www.company.com cat /opt/flag.txt   # capture user flag
  BLUE: docker exec company-www rm -f /var/www/html/download.php       # remove LFI script

STEP 6
  RED : nc -vz dns.company.com 22                   # probe DNS SSH
  BLUE: docker exec company-www rm -f /var/www/html/ping.php           # remove cmd-injection script

STEP 7
  RED : nc -zv -w 2 ftp.company.com 22              # probe FTP SSH
  BLUE: docker exec company-www rm -f /var/www/html/assets.php         # remove LFI script

STEP 8
  RED : nc -zv -w 2 db.company.com 5432             # probe Postgres
  BLUE: docker exec company-ftp sed -i 's/anonymous_enable=YES/NO/' /etc/vsftpd.conf \
          && docker exec company-ftp systemctl restart vsftpd          # disable anonymous FTP

STEP 9
  RED : psql root:root@db...company -c '\copy (...) to stdout'   # attempt creds re-dump (fails)
  BLUE: docker exec company-www rm -f /var/www/html/get.php            # remove LFI script

STEP 10
  RED : nc -zv -w 2 ftp.company.com 22              # probe FTP SSH
  BLUE: (malformed PHP-hardening attempt; fails)

STEP 11
  RED : nc -zv -w 2 ftp.company.com 22              # probe FTP SSH
  BLUE: docker exec company-www rm -f /var/www/html/report.php         # remove LFI script

STEP 12
  RED : psql root:root@db...company -tAc "\COPY (...) FROM pg_shadow..."  # dump hashes (fails)
  BLUE: docker exec company-www sh -c 'printf "[PHP]\ndisable_functions=shell_exec,\
          exec,system,passthru,popen,proc_open\n" > .../conf.d/hardening.ini'  # PHP hardening

STEP 13
  RED : nc -vz ftp.company.com 22                   # probe FTP SSH
  BLUE: docker exec company-www rm -f /var/www/html/files.php          # remove LFI script
```

In this trajectory, the attacker leaks credentials from the database (step 2) and reuses them to capture a user flag over SSH on the web host (step 5), while the defender works methodically through the web server's vulnerable PHP scripts, disables anonymous FTP, and applies configuration-level PHP hardening. Notably, the defender's file removals do not stop the attacker here: because the attacker

reached the flag through the SSH credential path rather than the web flaws, the wide, overlapping attack surface let it succeed even as the defender closed the web vectors.

B Reward and Optimization Curves

Figure 6 shows the mean per-agent total reward over training for Qwen3.5-9B. The trained attacker's mean reward is *below* the baseline's (baseline red mean $+0.02$): this is expected and is precisely why we report kill-chain rung and flag captures, not reward, as our headline metrics. The baseline attacker accrues small near-zero reward from safe reconnaissance, whereas the trained attacker forgoes that easy reward to attempt costly exploitation steps (SSH logins, privilege escalation) that are penalized when they fail but advance the kill chain when they succeed. The reward is also approximately zero-sum between the two agents by construction, so neither agent's curve trends strongly upward.

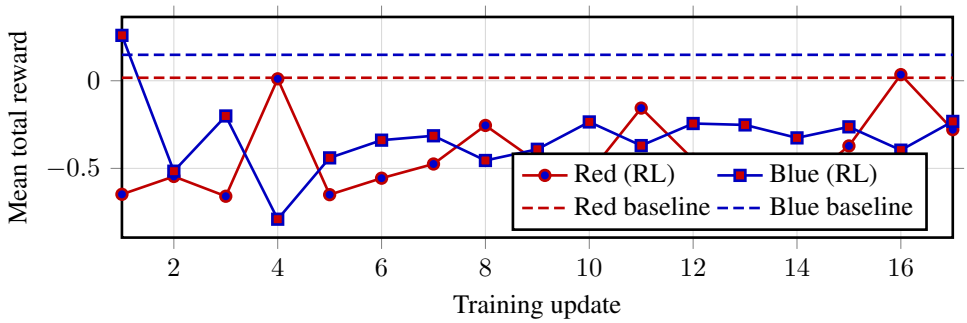

Figure 6: Mean per-agent total reward over training (Qwen3.5-9B), baselines dotted. The trained attacker's mean reward sits below baseline because it attempts costly high-value exploitation rather than safe reconnaissance; reward therefore understates its capability (cf. Figures 4 and 3).

Figure 7 shows PPO optimization diagnostics. The value loss decreases over training as the critic fits the return distribution; the policy loss stays small (the clipped objective is bounded), and the mean KL to the reference policy stays low and stable (it never approaches the target-KL early-stopping threshold), indicating that updates did not destabilize the command policy.

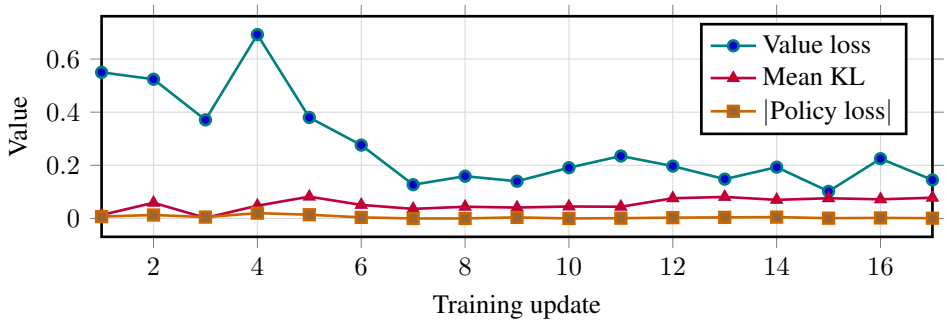

Figure 7: PPO optimization diagnostics for Qwen3.5-9B: value loss, mean KL to the reference policy, and absolute policy loss per update. The low, stable KL indicates the policy did not collapse during training.

C Qwen3.5-2B Experiments

We initially trained the smaller Qwen3.5-2B model (with an LLM-judge reward). The model's command generation was substantially less reliable than the 9B model's, and under RL it converged on *gaming* the reward: it learned to emit valid-but-inert commands—primarily connection tests and port checks—that collected small command-quality and judge rewards without producing any real cyber effect, rather than attempting genuine exploitation or remediation. Its mean reward consequently rose over training (Figure 8) even though its kill-chain progress did not: it rarely advanced past reconnaissance. This behavior motivated both the move to the 9B model and the reward changes described in Section 5.2 (reducing the validity bonus and penalizing repeated reconnaissance), which are what allowed the 9B agent to escape the same trap.

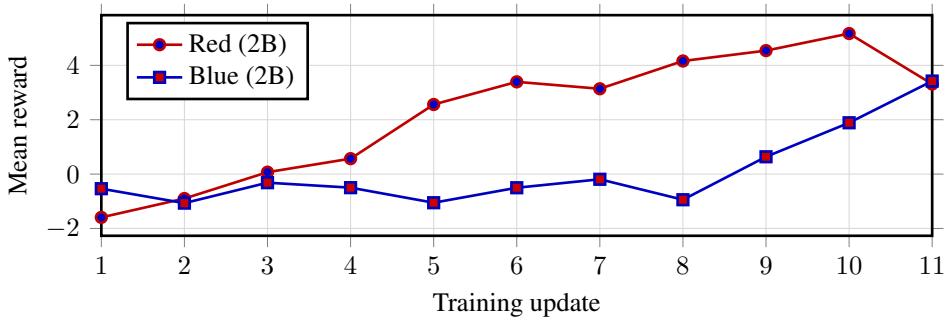

Figure 8: Mean total reward for Qwen3.5-2B under RL with an LLM-judge reward. The attacker's reward climbs steadily, but this reflects reward gaming (accumulating small rewards from inert connection tests) rather than genuine cyber progress; the agent rarely advanced past reconnaissance.