Moving Target Offense: RL-based Adaptive Evasion Strategies for C2 Frameworks

Karim Khamaisi[2], Anton Crazzolara[1], Aleksandar Ristic[1], Samuel Brügger[1],
Bruno Rodrigues[2], Jan von der Assen[1], Burkhard Stiller[1]
[1]Communication Systems Group CSG, Department of Informatics IfI, University of Zurich UZH, Switzerland
[2]Embedded Sensing Group ESG, School of Computer Science SCS, University of St. Gallen HSG, Switzerland
E-mail: [name.lastname]@uzh.ch, [vonderassen|stiller]@ifi.uzh.ch, bruno.rodrigues@unisg.ch

2025 IEEE 50th Conference on Local Computer Networks (LCN) | 979-8-3315-3703-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/LCN65610.2025.11146346

***Abstract*—Modern Intrusion Detection Systems (IDS) that rely on signature-based detection struggle to detect novel threats like AI-based Command-and-Control (C2) frameworks, highlighting a gap in the ability of current systems to detect and mitigate emerging botnet threats. This work addresses this gap by investigating NimPlant's detection and evasion capabilities.**

This paper (1) shows strategies to detect NimPlant bots using network traffic analysis, (2) enhances their evasion capabilities using Reinforcement Learning (RL), and (3) evaluates AI-driven evasion against IDS. Using a controlled testing environment, an infected device is simulated by implementing evasion strategies and integrating them into an RL-based AI system. Herewith, AI significantly improves evasion, with the RL-enhanced NimPlant achieving higher detection bypass rates. However, the IDS configuration heavily impacts AI effectiveness, underscoring the need for robust security setups. Thus, recommendations are provided for detecting bot infections and countering AI-enhanced botnets.

***Index Terms*—Botnet, Command-and-Control, Malware, Reinforcement Learning, Intrusion Detection**

I. Introduction

Within cybersecurity, the rapid evolution of Artificial Intelligence (AI) is a dual-edged sword. It allows for a fast-paced advance in both offensive and defensive capabilities. Although AI significantly improves threat detection on the defense side with modern Intrusion Detection Systems (IDS) relying on Deep Packet Inspection (DPI) [8], attackers have increasingly adopted AI to strengthen malware evasion and botnet resilience [9]. To manage these botnets, attackers rely on Command-and-Control (C2) frameworks, which provide a centralized platform for issuing commands, collecting data, and updating malware to infect additional devices [10]. Detecting C2 activity is critical for mitigating threats, as it enables security teams to identify and respond to malicious behavior promptly.

AI-powered malware represents the next evolutionary step in this arms race, capable of dynamically adjusting behaviors to avoid detection while seamlessly blending malicious activity with normal traffic patterns. For example, Reinforcement Learning (RL) techniques have been used in ransomware [20] to adapt their encryption behavior and stay stealthy while encrypting files. Similarly, C2 frameworks can leverage RL and Generative Adversarial Networks (GAN) to hide malicious communication patterns, simulate legitimate network traffic, and evade IDS or Intrusion Prevention System (IPS) [2], [4], [15]. These AI-driven approaches enable malware to learn and adapt to defensive measures in real-time, significantly complicating detection efforts.

Despite the growing threat posed by AI-powered botnets, a critical gap remains in the literature regarding offensive applications of AI to enhance botnet evasion policies. Current research primarily focuses on static evasion techniques or binary-level obfuscation, leaving network-level adaptive strategies underexplored. This paper addresses this gap by investigating how AI-driven decision-making can enhance the resilience and evasion capabilities of C2 frameworks through dynamic strategy optimization. Based on the NimPlant C2 framework [19], [18], this paper explores RL's potential to dynamically optimize botnet evasion policies against known IDS/IPS defenses. The research contributes to understanding the offensive applications of AI in cybersecurity, while providing insights for developing more robust defensive systems. Our contributions include:

- RL framework for evasive C2 communication: a Q-Learning-based agent dynamically selects and combines up to 7 evasion strategies during runtime. The Snort network IDS was employed and configured with both naive and expert rule sets.
- Quantitative evaluation in realistic detection scenarios: experiments demonstrate RL agent convergence in 1.5 and 5.7 hours against naive and expert rules respectively, achieving alert-free communication with minimal active strategies.
- Strategic behavior profiling via heatmap analysis: temporal heatmaps reveal that the agent learns minimal yet effective evasive configurations—such as port hopping and endpoint rotation—under naive rules, while expert rules prompt broader and more adaptive combinations across episodes.

The remainder of the paper is organized as follows. Section II discusses related work, addressing AI-based evasion techniques. Section III introduces the design and implementation, followed by the evaluation of experiments in Section IV. Section V discusses the results of the experiment and presents the limitations of this work. Finally, conclusions and future work are part of Section VI.

II. Related Work

Recent large-scale studies show that hybrid systems—signature plus anomaly or flow analytics—outperform either technique in isolation [17]. IBM introduced DeepLocker as a proof-of-concept that combines AI with traditional obfuscation methods to embed malware within benign applications, revealing its payload when specific target conditions, such as facial features or location, are met [11], [12]. This makes detection and reverse engineering extremely difficult due to the opaqueness of deep learning models.

RL has also been applied for evasion. An RL agent was trained to mutate Portable Executable (PE) malware samples by applying operations like header modifications and section appends, with the goal of evading a static ML-based antivirus engine [1]. Results showed a notable drop in VirusTotal detection rates—from 52.5/65 to 16.5/65—after RL-guided transformations. Advanced RL frameworks using Shapley value guidance for action prioritization have achieved 76.88% evasion rates on EMBER datasets [21].

Another approach uses GANs to adapt malware's network behavior. Rigaki and Garcia [15] modified a Remote Access Trojan (RAT) to mimic Facebook chat traffic using GANs, enabling the malware to bypass a behavioral IPS. The GAN iteratively trained using feedback from the IPS to refine its mimicry until it was no longer blocked.

Unlike prior GAN-based traffic morphing [6] or PE-mutation agents that focus on binary-level evasion [7], this paper presents the first RL framework for network-level C2 evasion in a real implant. By operating at the packet-and-header layer, it complements malware-centric approaches and directly evaluates detection logic designed for network IDS.

III. RL-enabled NimPlant

The NimPlant code was modified to dynamically change strategies in two cases: when triggered by the botmaster in a proactive manner and when triggered by the RL agent. Fig. 1 shows the NimPlant RL architecture, illustrating key components and their interactions. At its core, a Q-Learning agent receives rewards based on evasion strategy configurations that successfully bypass the IDS. The RL agent tracks alerts through a counter mechanism synchronizing with IDS alert logs. Further, the target communicates with the attacker's C2 server via HTTP while the IDS monitors this traffic and generates alerts based on rule-based detection.

Snort was configured to alert based on the rules presented in TABLE I. The design of these rules is based on the scenario, the assumptions, and the indicators of compromise. The rules are divided into naive rules and expert rules, following the assumptions mentioned previously.

A. Command-based Evasion Strategies

After the configuration of the rules, the initial NimPlant version could be detected and alerted by Snort. Thus, the NimPlant code was adapted at this stage to change strategies if proactively commanded by the botmaster. TABLE II describes the strategies that were implemented and could also be enabled at runtime when giving commands to the server using the console:

TABLE I: Snort Rules Overview

Model	*Rule Type*	*Description*
Naive	Keyword: "nimplant"	Alerts on suspicious keywords in packet content.
	Keyword: "register"	Commonly used in registration or setup phases of malware.
	Keyword: "task"	May indicate command assignment in malware communications.
	Keyword: "result"	Often used to transmit execution results back to the attacker.
	Specific ports	Limits alerts to high-risk ports.
Expert	All ports	Covers attacks on any port.
	Host header	Alerts on uncommon host values (*e.g.*, IPs).
	Packet freq. & size	Detects patterns like DDoS or keep-alive.

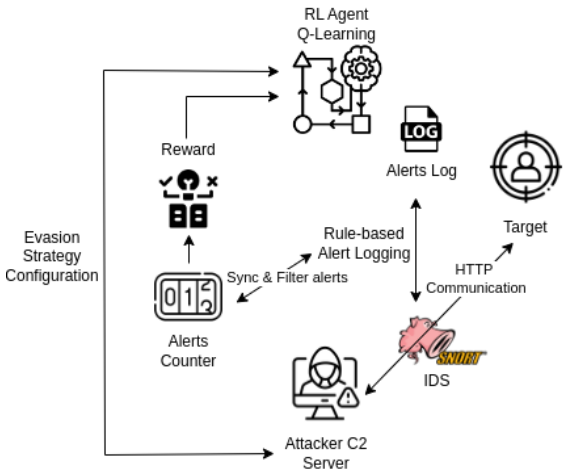


Fig. 1: NimPlant RL Architecture

TABLE II: Evasion Strategies

Nr.	*Strategy*	*Description*
1	Server name change	Random server names (*e.g.*, Apache, GWS) after 100 requests.
2	User agent change	Client changes user agent after 50 requests.
3	Port hopping	Communication ports change every 10 minutes.
4	Endpoint change	Endpoints vary during communication.
5	Host header change	IP in header replaced with random web page.
6	Frequency change	Command request intervals vary over time.
7	Packet size change	Varies packet sizes to avoid detection.

Enabling or disabling any subset of $\{S1,\ldots,S7\}$ at each polling interval gives the attacker fine-grained control over observable traffic features. This control surface is the basis for the RL training described in Section III, where the agent learns which combination of strategies minimizes Snort alerts under naive and expert rule sets.

B. Adaptive Policies via RL

To develop evasive behaviors against Snort, Q-Learning is used to train an agent that dynamically selects strategies to hide C2 communication patterns between the NimPlant server and clients. Each state $s \in \mathcal{S}$ is a 7-bit vector indicating the on/off status of evasion strategies (*e.g.*, changing endpoints, altering packet sizes, keyword obfuscation). This yields $|\mathcal{S}| = 2^7 = 128$ distinct configurations.

The agent's action space $\mathscr{A}$ consists of 8 discrete actions: enabling one of the 7 strategies or doing nothing (*i.e.*, also an action to remain stealth). These actions represent tactical adjustments made by the botmaster to reduce the visibility of malicious traffic. The Q-table $Q(s,a)$, of shape 128×8, maps each state-action pair to a value indicating the expected long-term reward of executing a in s.

The training begins with random Q-values:

$$Q(s,a) \sim \mathscr{U}(0, 0.01)$$

At each timestep, an action a is chosen via an ϵ-greedy policy to balance exploration and exploitation by the agent:

$$a = \begin{cases} \text{random action}, & \text{with probability } \epsilon = 0.2 \\ \arg\max_{a'} Q(s,a'), & \text{otherwise} \end{cases}$$

After executing a, the environment returns the next state s', a scalar reward r, and a done flag. The Q-table is then updated using the Bellman equation as follows:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') \right]$$

This paper uses $\alpha = 0.2$ to moderately incorporate new information, and $\gamma = 0.95$ to prioritize future alert reduction. Each episode begins with all strategies disabled and terminates once no alerts are triggered and at most five strategies are active.

The reward function is used to penalize alerts proportionally to their severity. Let A be the set of triggered Snort alert SIDs, and $w(a_i)$ their weights ($1 \le w(a_i) \le 5$). Let k be the number of enabled strategies. Then:

$$r = \begin{cases} 10 - k, & \text{if } A = \emptyset \\ -4 \cdot \sum_{a_i \in A} w(a_i) - k, & \text{otherwise} \end{cases}$$

This is used to foster stealthy configurations with a minimal alert generation and minimal strategic overhead as possible. The constants 10 and -4 were chosen to balance exploration and alert minimization: +10 provides sufficient positive reinforcement for alert-free states while encouraging strategy efficiency through the -k penalty, while -4 ensures meaningful punishment even for low-weight alerts (minimum -4 for weight=1) and substantial deterrence for critical alerts (up to -20 for weight=5). After training, the optimal evasion policy $\pi(s)$ is derived by selecting the highest-valued action for each state:

$$\pi(s) = \arg\max_{a} Q(s,a)$$

All agent interactions, state transitions, actions, alert counts, and rewards, are logged. A fail-safe mechanism is then used to store the Q-table and episode index after each episode, ensuring the possibility to resume and trace outcomes of the training.

IV. Experimental Evaluation

A. Setup

The client environment consisted of a Windows 10 desktop system with network traffic monitored via Wireshark, while the server operated on an Ubuntu 23 virtual machine hosting the NimPlant C2 framework with necessary dependencies installed.

The experimental configuration enabled comprehensive analysis of communication patterns between infected clients and the command server. NimPlant employs XOR-based encryption over HTTP, resulting in obfuscated payload data visible in network captures as shown in Fig. 2. Analyzing the communication in Wireshark allows for uncovering indicators of compromise as marked green. Data of the infected system processes was collected using Sysmon to establish baseline measurements before and after the infection. Additionally, the whole experiment was screen-recorded.

Fig. 2: Indicators of Compromise (IoC) in the communication channel

B. Network Analysis

NimPlant uses REST APIs with GET/POST requests for bot coordination. The compromised client initiates contact via GET requests, receives an identification token, then completes registration via POST request with encrypted victim data.

C. NimPlant with Ransomware

A ransomware scenario was chosen as plausible for real-world attacks, where hackers employ C2 frameworks like NimPlant as first-stage infection vectors before deploying encryption malware. For this analysis, the Crypter [1] ransomware was selected as an open-source educational tool developed by researchers for experimental purposes.

[1] https://github.com/m0n0ph1/Crypter-2

NimPlant successfully transferred and executed Crypter on the target machine, demonstrating how the framework disrupts normal communication patterns through variable-length packets during file operations. While real-world deployment would require additional binary obfuscation techniques [14], this confirms NimPlant's capability as an effective attack vector.

D. Learning Strategies

As shown in TABLE III, the model trained against naive Snort rules converged significantly faster than the one trained against expert rules. The naive model completed 100 episodes in roughly 1 h, whereas the expert model required approximately 6 h. This disparity stems from the increased complexity of the expert rule set, which imposes more challenging evasion conditions and extends the learning process.

TABLE III: Training time for expert vs. naive models

Episode Group	*Expert (hrs)*	*Naive (hrs)*
1 (Episodes 1–20)	2.1	0.2
2 (Episodes 21–40)	1.2	0.2
3 (Episodes 41–60)	1.0	0.2
4 (Episodes 61–80)	0.7	0.2
5 (Episodes 81–100)	0.7	0.2
All episodes (Total)	5.7	1.0

The table also presents training time grouped by 20-episode intervals. The expert model shows a clear downward trend in training duration across groups, indicating that the RL agent gradually learned to exploit effective strategies and reduce alerts. In contrast, the naive model maintained low and stable training times across all groups, reflecting the relative simplicity of the environment and the early discovery of optimal evasion strategies.

E. Q-Learning Results — Actions and Rewards

Fig. 3 shows the number of actions taken per episode under expert and naive rule sets. Both models initially perform more actions due to exploration. Over time, the naive model quickly stabilizes as it identifies effective strategies, while the expert model exhibits fluctuations due to more complex exploration and exploitation dynamics.

Fig. 4 presents the corresponding reward evolution. The expert model shows negative spikes early on, reflecting suboptimal strategy combinations. From episode 60 onward, the rewards improve as the model exploits learned strategies. Despite negative rewards, episodes may still terminate early if alert thresholds are met. A reward counter—triggering completion after five positive rewards or one exceeding 4, ensures progression and stability.

These results confirm that the RL agent adapts dynamically to the detection environment. Rapid stabilization under naive rules indicates low exploration cost and fast policy convergence. In contrast, persistent fluctuations under expert rules highlight the need for sustained exploration due to overlapping or ambiguous detection criteria.

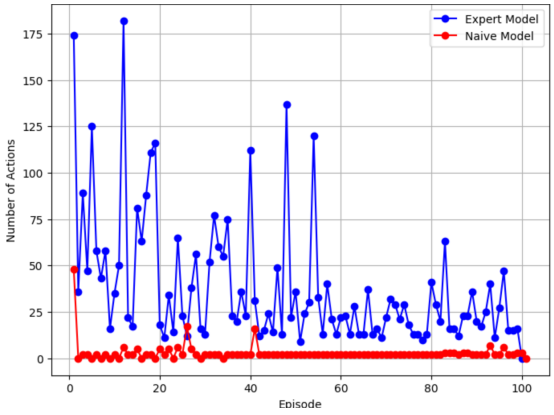


Fig. 3: Number of actions per episode

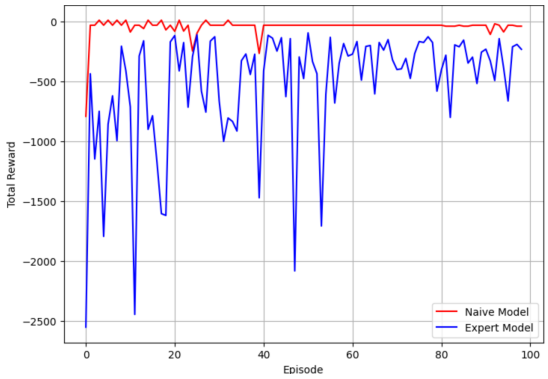


Fig. 4: Amount of reward per episode

F. Q-Learning Results - Strategies

Fig. 5a presents a heatmap illustrating strategy counts operating against expert Snort rules across 100 episodes. Initially, the heatmap shows dense, dark areas, indicating increased strategy enablement in the early training phases.

Fig. 5b replicates this heatmap for the model under naive rules. In contrast to expert rules, fewer dense dark areas indicate that the model sufficiently explores the set of optimal strategies without requiring extensive exploration.

Fig. 6a and 6b introduce the normalized and grouped counts of enabled strategies by the RL approach, contrasting with the absolute counts depicted in the prior heatmaps.

Normalization is used to scale strategy counts relative to the total actions taken within each episode group. This eliminates potential biases from variations in the total number of actions across the different episode groups. Grouping counts over 20 episodes provide a condensed overview, smoothing specific anomalies of episodes to view general trends, and helping to identify consistent patterns of effective strategies in the training. It provides a more precise narrative of the model's strategic evolution, especially in distinguishing recurrent and impactful strategies from sporadic or context-specific occurrences.

Fig. 6a presents normalized values for strategy counts grouped over 20 episodes. The dark areas, excluding strategy

TABLE IV: Normalized strategy usage by 20-episode groups with aggregate use.

Model	Strategy	Grp 1	Grp 2	Grp 3	Grp 4	Grp 5	Corr TT	Corr Block	Agg. Usage	Peak Grp
Expert	Server name change (S1)	14.60	15.69	11.94	14.57	21.68	−0.37	−0.79	**78.48**	5
	User agent change (S2)	15.61	20.99	15.92	14.78	12.11	+0.23	−0.64	79.41	2
	Port hopping (S3)	13.95	9.71	9.50	7.49	9.38	+0.96	+0.84	50.03	1
	Endpoint change (S4)	13.06	11.40	14.76	15.38	15.82	−0.53	+0.39	70.42	5
	Host header change (S5)	12.64	11.40	16.82	23.89	18.55	−0.67	−0.07	**83.30**	4
	Frequency change (S6)	14.12	14.90	12.71	9.31	10.16	+0.70	−0.42	61.20	2
	Packet size change (S7)	12.82	12.75	14.89	13.36	10.94	+0.16	−0.42	64.76	3
	No strategy (S0)	3.20	3.16	3.47	1.21	1.37	+0.67	−0.67	12.41	3
	Training time (h)	**2.1**	**1.1**	**1.1**	**0.7**	**0.7**	–	–	–	–
Naive	Server name change (S1)	14.42	12.12	0.00	0.00	3.39	−0.37	−0.79	29.93	1
	User agent change (S2)	5.77	11.11	0.00	0.00	1.69	+0.23	−0.64	18.57	2
	Port hopping (S3)	18.27	18.18	33.33	33.33	32.20	+0.96	+0.84	135.31	3
	Endpoint change (S4)	25.00	35.35	66.67	66.67	33.90	−0.53	+0.39	**227.59**	3
	Host header change (S5)	15.38	0.00	0.00	0.00	13.56	−0.67	−0.07	28.94	1
	Frequency change (S6)	13.46	10.10	0.00	0.00	10.17	+0.70	−0.42	33.73	1
	Packet size change (S7)	3.85	10.10	0.00	0.00	3.39	+0.16	−0.42	17.34	2
	No strategy (S0)	3.85	3.03	0.00	0.00	1.69	+0.67	−0.67	8.57	1
	Training time (h)	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	–	–	–	–

0, imply substantial counts across various strategies, outlining the absence of a dominant strategy to evade expert rules. For instance, Strategy 1 (related to the NimPlant keyword) and Strategy 5 exhibit consistent dark areas, highlighting their significance.

Fig. 6b shows the model operating under naive rules. Strategies 3 and 4 are notably prevalent in groups 3 and 4 of episodes. Strategy 3, which utilizes port hopping, and Strategy 4, involving changes in communication endpoints, are identified as the most effective evasive techniques. Strategy 3 supports the paper's hypothesis that expert security analysts consistently block traffic on default ports (*e.g.*, 23/TCP for Telnet or 445/TCP for SMB), rendering port hopping a viable evasion method.

TABLE IV complements the normalized heatmaps in Fig. 6 by listing the exact per-group shares, two correlation metrics, and a cumulative–usage column.

For the *expert* rules, port hopping (S3) correlates strongly with longer convergence (r=0.96), while host-header disguise (S5) shortens it ($r = -0.67$). S5 is also the most used tactic overall (83% aggregate), whereas S3, though influential early, contributes only 50%. The other strategies have moderate correlations, reflecting their varying impacts on convergence. Peak-group data show endpoint change (S4) maximizing in Group 5, matching the broader tactic mix seen late in training.

With the *naive* rules, port hopping and endpoint change dominate: S3 usage rises with block index (r=0.84) and, together with S4, accounts for ≈63% of all actions. Low-value tactics (packet-size padding, "no strategy") have negative correlations ($r \leq -0.42$) and minimal share. Training time stays constant at 0.2 h, confirming that simple signature rules are bypassed once a small, high-reward subset is found.

G. Rewards per Episode Group

Fig. 7 show the distribution of per-episode rewards for both the naive and expert rule sets, divided into five consecutive intervals of twenty episodes each.

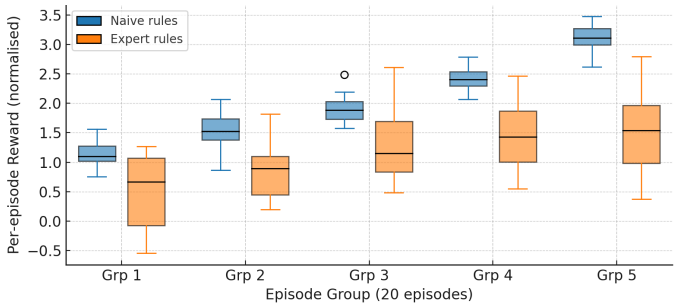
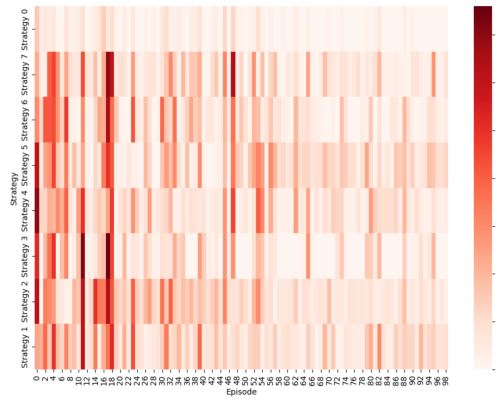


Fig. 7: Per-episode reward distributions for naive and expert rule sets, grouped into five 20-episode intervals.

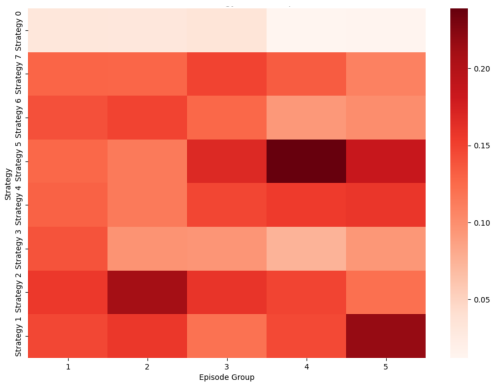
Under the expert rule set, the first two groups exhibit wide inter-quartile ranges and numerous outliers, indicating high variability as the agent explores a more complex evasion space. The median reward in Group 1 is close to zero, reflecting frequent alert-triggering actions, and only by Group 4 does the median approach positive values consistently. The naive agent achieves higher median rewards and narrower IQRs from Group 2 onward, showing stable exploitation of effective strategies. Both agents reduce variance by Group 5, but the expert agent retains a heavier tail, indicating occasional suboptimal regressions. The reward distribution analysis reveals that expert rule evasion requires coordinated multi-strategy activation rather than individual techniques. Future C2 frameworks may need sophisticated coordination mechanisms to evade advanced detection systems.



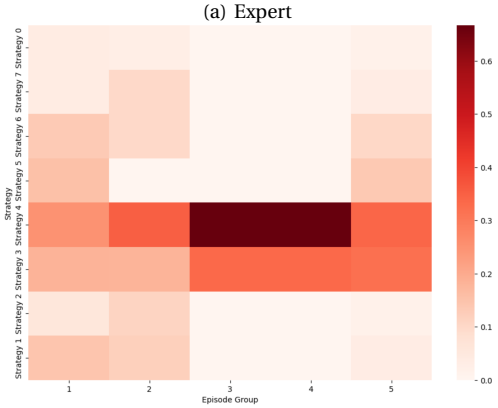
(a) Expert

(b) Naive

Fig. 5: Strategy usage heatmaps across expert and naive Snort rule sets — Absolute counts over 100 episodes



(a) Expert



(b) Naive

Fig. 6: Normalized strategy usage heatmaps across expert and naive. Grouped by 20-episode intervals.

V. Discussion and Limitations

We show that RL–driven evasion of network-based IDS becomes substantially more difficult as rule-set complexity increases: under naive Snort rules the Q-Learning agent converges in approximately 1 h with fewer than 750 cumulative alerts, whereas under expert rules convergence requires ≈6 h and ≈4,500 alerts (TABLE III). Strategy-usage heatmaps show rapid specialization on port hopping (S3) and endpoint rotation (S4) in the naive regime, while expert rules force a diversified tactic mix and yield high reward variance in early exploration (Fig. 6, 7).

The agent's adaptation speed depends not only on the polling frequency (*e.g.*, the NimPlant *sleepTime* interval), but also on the time required to decide which strategy to toggle. For instance, if a polling interval is 5 s, any decision delay by humans or systems longer than 5 s removes the advantage of quicker polling; however, decision loops under 5 s fully exploit it. Conversely, shortening *sleepTime* for greater agility may trigger detection risks, as frequent GET/POST bursts can mimic DDoS attacks or unusual activities.

The observed convergence patterns demonstrate fundamental differences in learning complexity between rule configurations. Expert rules necessitate prolonged exploration with persistent reward variance, highlighting the need for adaptive coordination mechanisms.

As attackers increasingly employ AI-driven evasion, the following multi-layered defense strategies are recommended [5], [13]:

- **Hybrid, automated rule adaptation:** deploy ML-based rule synthesis to continuously update signature and behavior rules in real time, preventing static policy reverse-engineering [3].
- **Adversarial training:** augment IDS with adversarial examples to harden classifiers [6], [7].
- **Ensemble detection:** combine multiple ML detector types for improved resilience [3].
- **Collaborative threat intelligence:** share adversarial signatures across organizations [16].

VI. Summary and Future Work

This paper presents a RL-based C2 framework that embeds seven discrete evasion strategies into the NimPlant implant and quantifies how rule complexity shapes evasive performance. Future work will train and evaluate the same agent on a simulated TLS channel—with handshake and payload-padding strategies—to measure changes in convergence time, alert exposure, and strategy selection under cryptographic overhead.

REFERENCES

[1] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth, "Learning to evade static pe machine learning malware models via reinforcement learning," *arXiv preprint arXiv:1801.08917*, 2018.

[2] H. S. Anderson, A. Kharkar, B. Filar, and P. Roth, "Evading machine learning malware detection," 2017.

[3] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, Vol. 18, No. 2, pp. 1153–1176, 2016.

[4] L. Chen, Y. Ye, and T. Bourlai, "Adversarial machine learning in malware detection: Arms race between evasion attack and defense," in *2017 European Intelligence and Security Informatics Conference (EISIC)*, 2017, pp. 99–106.

[5] Cloud Security Alliance, "How is ai changing cybersecurity? 6 key insights from csa's 2024 state of ai and security survey report," Web article, January 2024. [Online]. Available: https://cloudsecurityalliance.org/articles/embracing-ai-in-cybersecurity-6-key-insights-from-csa-s-2024-state-of-ai-and-security-survey-report

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems*, Vol. 27, 2014.

[7] L. Huang, A. D. Joseph, B. Nelson, B. I. P. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, 2011, pp. 43–58.

[8] Đ. D. Jovanović and P. V. Vuletić, "Analysis and characterization of iot malware command and control communication," in *2019 27th Telecommunications Forum (TELFOR)*. IEEE, 2019, pp. 1–4.

[9] N. Kaloudi and J. Li, "The ai-based cyber threat landscape: A survey," *ACM Computing Surveys (CSUR)*, Vol. 53, No. 1, pp. 1–34, 2020.

[10] S. Khattak, N. R. Ramay, K. R. Khan, A. A. Syed, and S. A. Khayam, "A taxonomy of botnet behavior, detection, and defense," *IEEE Communications Surveys & Tutorials*, Vol. 16, No. 2, pp. 898–924, 2014. [Online]. Available: https://doi.org/10.1109/SURV.2013.091213.00134

[11] D. Kirat, J. Jang, and M. Stoecklin, "Deeplocker–concealing targeted attacks with ai locksmithing," 2018. [Online]. Available: https://i.blackhat.com/us-18/Thu-August-9/us-18-Kirat-DeepLocker-Concealing-Targeted-Attacks-with-AI-Locksmithing.pdf

[12] Kirat, Dhilung and Jang, Jiyong and Stoecklin, Marc, "Deeplocker: How ai can power a stealthy new breed of malware," 2018. [Online]. Available: https://securityintelligence.com/deeplocker-how-ai-can-power-a-stealthy-new-breed-of-malware/

[13] OpenText Cybersecurity, "Opentext cybersecurity 2024 global ransomware survey: Supply chain and ai-powered attack fears intensify," Blog post, June 2024. [Online]. Available: https://blogs.opentext.com/opentext-cybersecurity-2024-global-ransomware-survey-supply-chain-and-ai-powered-attack-fears-intensify/

[14] I. V. Popov, S. K. Debray, and G. R. Andrews, "Binary obfuscation using signals." in *USENIX Security Symposium*, 2007, pp. 275–290.

[15] M. Rigaki and S. Garcia, "Bringing a gan to a knife-fight: Adapting malware communication to avoid detection," in *2018 IEEE Security and Privacy Workshops (SPW)*, 2018, pp. 70–75. [Online]. Available: https://doi.org/10.1109/SPW.2018.00019

[16] C. Sauerwein, C. Sillaber, A. Mussmann, and R. Breu, "Threat intelligence sharing platforms: An exploratory study of software vendors and research perspectives," 2017.

[17] R. Tong, M. Lu, Z. Zhang, and W. Lee, "A systematic evaluation of hybrid intrusion detection systems," in *Network and Distributed System Security Symposium (NDSS)*, 2022.

[18] C. Van Cooten, "Building a c2 implant in nim - considerations and lessons learned." 2021. [Online]. Available: https://casvancooten.com/posts/2021/08/building-a-c2-implant-in-nim-considerations-and-lessons-learned/#nim-for-offensive-security

[19] C. Van Cooten, F. Mosch, F. Göksel, K. Yamamoto, and M. Velazco, "Nimplant - a light firststage c2 implant written in nim and python." GitHub. [Online]. Available: https://github.com/chvancooten/NimPlant

[20] J. von der Assen, A. H. Celdrán, J. Luechinger, P. M. S. Sánchez, G. Bovet, G. M. Pérez, and B. Stiller, "Ransomai: Ai-powered ransomware for stealthy encryption," in *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE, 2023, pp. 2578–2583.

[21] D. Zhan, W. Bai, X. Liu, Y. Hu, L. Zhang, S. Guo, and Z. Pan, "Psp-mal: Evading malware detection via prioritized experience-based reinforcement learning with shapley prior," in *Proceedings of the 39th Annual Computer Security Applications Conference*, 2023, pp. 580–593.