AI-Based Attacks and Defenses on Intrusion Detection Systems

Sergio Valderrama (sevalder@ucsc.edu)
University of California, Santa Cruz

Russell Elliott (rdelliot@ucsc.edu)
University of California, Santa Cruz

Abstract

In the evolving landscape of cybersecurity, artificial intelligence is being leveraged not just for defense, but also to test and improve offensive capabilities. This presentation explores an innovative approach to brute force attack testing using AI agents in various simulated environments. We'll delve into the testing setups, agent behaviors, key findings, and the implications for future cybersecurity practices. By pushing the boundaries of AI in penetration testing, we aim to uncover both the potential and limitations of these advanced techniques in real-world scenarios.

1 Introduction

The use of artificial intelligence (AI) in cybersecurity is growing rapidly. While AI can be used for defense, it is also being utilized to test and improve offensive capabilities. We will analyze an innovative approach to brute force attack testing using AI agents in various simulated environments. Traditionally, brute force attacks involved systematically trying every possible password combination. However, with the advent of AI, these attacks can be conducted much more effectively. AI agents can learn and adapt to different security measures, making them more difficult to detect and block. We detail the architecture and methodology used in this research, including the four different testing environments and AI models and actions.

We would talk about the challenges and solutions encountered during AI training, such as overemphasis on waiting, lack of focus on attempts, and environment reset delays. By addressing these challenges, we were able to shape the AI agents' behavior to more closely mimic real-world attack patterns.

The results of the research demonstrate the effectiveness of AI agents in conducting brute force attacks. The agents were able to successfully breach three of the four testing environments, even those with sophisticated security measures. This article provides valuable insights into the potential and limitations of AI in complex attacks. By understanding the tactics employed by AI-powered adversaries, security researchers can design countermeasures that can anticipate and mitigate future threats.

2 Related Work

The intersection of artificial intelligence and cybersecurity has given rise to a new era of adversarial techniques. Reinforcement learning (RL), in particular, has emerged as a powerful tool for developing autonomous agents capable of circumventing security defenses. Recent research has demonstrated the efficacy of RL-based approaches in evading web application firewalls (WAFs), intrusion detection systems (IDSs), and malware detection systems.

Early works in this domain explored the feasibility of using RL to craft malicious payloads that could bypass signature-based WAFs. For instance, [1–3, 8] employed techniques like Deep Q-Learning (DQN) and Proximal Policy Optimization (PPO) to generate evasive payloads. While these studies provided foundational insights, they often relied on simplified WAF models or limited attack scenarios.

More recent research has delved into the complexities of real-world security environments. [4, 7, 9] have shown that RL agents can effectively adapt to dynamic defense mechanisms, such as machine learning-based WAFs and malware detectors. By iteratively learning from interactions with these systems, these agents can discover subtle vulnerabilities and exploit them to launch successful attacks.

The adversarial nature of cybersecurity has also inspired the development of innovative RL-based frameworks. [8] proposed a novel approach that trains both an attacker and a defender agent simultaneously, simulating a competitive environment. This zero-sum game fosters the development of increasingly sophisticated attack and defense strategies.

While these advancements in offensive AI pose significant challenges to security practitioners, they also provide valuable insights for developing more robust defense mechanisms. By understanding the tactics employed by AI-powered adver-

saries, security researchers can design countermeasures that can anticipate and mitigate future threats.

3 Architecture and Methodology

This section describes the framework designed to train AI agents for brute force attack testing. The core objective is to provide a platform where agents can learn optimal strategies while navigating various security measures. The framework employs reinforcement learning models, allowing agents to adjust attack parameters iteratively to bypass defenses.

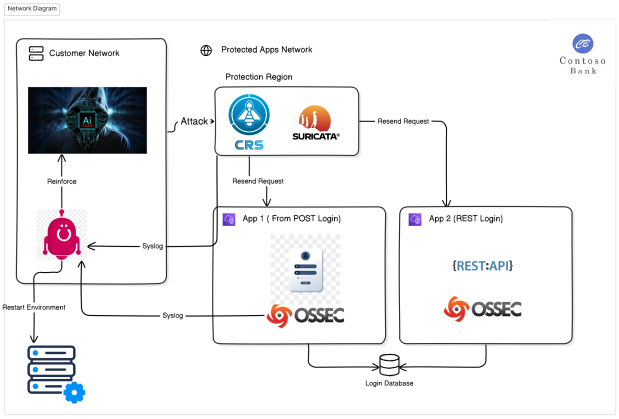

Figure 1: Enter Caption

3.1 AI Model: Intelligent Hacking Auto Attacking Agent (IH3A)

The IH3A agent is the heart of our framework. This sophisticated agent is designed to execute attacks based on predefined parameters while continuously learning and adjusting its strategies in response to defensive measures. The key aspects of the IH3A agent are:

- **Multi-step attack analysis**: The agent is able to perform multi-step attacks, bringing a novel way to perform attacks different from single-step attacks research so far.
- **Parameter Initialization**: The agent starts with predefined attack parameters based on initial training data and environment.
- **Attack Execution**: The IH3A agent launches attacks against the simulated network, monitoring defensive responses in a non-stationary reward environment. [6]
- **Feedback Analysis**: The agent processes feedback from the RLHelper (discussed below), analyzing detection patterns and attack outcomes. This helps us create a single pipeline for multiple non-stationary reward mechanisms.
- **Strategy Adjustment**: Based on feedback analysis, the IH3A agent dynamically adjusts its attack parameters to improve evasion and success rates.

The agent's performance is measured using a scoring system that considers: the number of attacks executed before detection, attack success rate, and attack duration. This ensures that we not only evade detection but also optimize for efficiency and effectiveness.

The strategies the agent can use during an attack:

- **Change IP**: The agent modifies its IP address to evade detection, which is particularly effective against IDS/WAF systems.
- **Adjust Attack Method**: It switches between password spraying, where multiple users are targeted with a common password, and traditional brute force methods, focusing on a single user with various passwords. This adaptability allows the agent to overcome diverse security measures.
- **Attempt Submission**: The agent sends login attempts. The agent is strongly biased to this action.
- **Strategic Waiting**: The agent pauses between attempts to avoid triggering rate-limiting mechanisms.
- **User Skipping**: The agent moves strategically to the next user on its list, distributing attempts to minimize the risk of locking accounts due to repeated failed attempts.

These actions are carefully orchestrated and prioritized by the agent to optimize the effectiveness of brute force attacks while minimizing the chances of being detected and blocked. The agent continuously refines its strategies through feedback and analysis, adapting to the specific challenges posed by each target environment.

3.2 RLHelper: Bridging the Agent and the Environment

The RLHelper component acts as a bridge between our IH3A agent and the simulated network environment. This helper system facilitates seamless communication and data management during the training process. The RLHelper consists of two primary applications:

- **Management API**: This API maintains the integrity of the training environment by providing functionality to:
  - Reset the database
  - Update data to prevent the agent from exploiting learned patterns
  - Restart web services as needed

- **SyslogServer**: A custom-built solution for high-speed, low-latency communication with our IH3A agent using shared memory.

The RLHelper provides several key functions to support our AI agent's training:

- **Dynamic Database Management**: Ensures fresh, realistic data for each training iteration.
- **High-Speed Synchronization**: Facilitates immediate communication between defensive systems and the IH3A agent.
- **Intelligent Password Generation**: Creates diverse, context-aware passwords to simulate real-world user behavior.
- **Syslog and API Metrics Gathering**: Collects and analyzes agent performance data for optimization and evaluation.

3.3 Testing Environments

The testing environments we used in our research are designed to simulate real-world scenarios with varying levels of security. They serve as a diverse playground for the AI agent to learn and adapt its brute force attack strategies. The environments are categorized as follows:

- **Env1-Lax**: This environment represents a basic, unsecured setup with no detection or blocking rules. It serves as a baseline to assess the AI agent's fundamental brute force capabilities without any security hurdles.
- **Env2-Strict**: This environment implements blocking mechanisms after a certain number of failed login attempts within a specific time window. It is based on a real-world implementation from a Latin American bank, using a JSON REST API for authentication. Notably, this environment uses user-based security, meaning blocking is tied to specific user accounts rather than entire sessions.
- **Env3-Stricter**: Env3 presents an even greater challenge with permanent blocking for failed login attempts. Authentication utilizes HTTP forms, similar to Env1. This combination of permanent blocking and a traditional authentication method tests the AI agent's adaptability and resilience against restrictive security measures.
- **Env4-FTP**: Shifting the focus from web applications to FTP servers, this environment uses a vanilla implementation of VSFTPd, relying on standard OS login procedures and local user accounts. This environment assesses the AI agent's ability to perform brute force attacks against a different type of service.

In addition to these core environments, our research delves into the application of a simulated defensive environment to elevate the training process. This environment is designed to emulate the essential principles of brute force protection mechanisms within complex architectures, utilizing network defense mechanisms as middleware, but faces challenges in accurately simulating real-world timing and complexities, especially those with advanced security measures.

3.4 Middleware Protection and Training Challenges

This section discusses the use of middleware for protection and the challenges faced during AI agent training.

3.5 Middleware Protection

Our research utilizes various middleware solutions to create a realistic defensive environment for testing AI-driven brute force attacks. These tools provide different layers of security and offer valuable feedback for the agent's learning process. The chosen middleware solutions include:

- **Suricata**: A high-performance Network Intrusion Detection System (NIDS) capable of real-time intrusion detection, inline intrusion prevention, and network security monitoring. Customized rules were created for Suricata to detect potential brute force attacks.
- **ModSecurity CRS**: A Web Application Firewall (WAF) providing protection against various web application attacks, including SQL injection and cross-site scripting. ModSecurity was configured with a CRS rule-set and further customized with strict rules for brute force detection. However, experimental rules proved largely ineffective.
- **OSSEC HIDS**: A Host-based Intrusion Detection System (HIDS) offering comprehensive log analysis, file integrity checking, and real-time threat detection. While the vanilla implementation of OSSEC does not include specific rules for brute force attacks, we implemented customized rules for our research.

3.6 AI Training Challenges and Solutions

Training the AI agent to effectively conduct brute force attacks presented several challenges. We highlight some of these challenges and the solutions we implemented to overcome them:

- **Overemphasis on Waiting**: The AI agent initially tended to favor waiting actions, potentially due to a lack of sufficient penalties for inaction. To address this, we heavily penalized waiting actions in the reward function, encouraging the agent to take more active steps.

- **Lack of Focus on Attempts**: To incentivize the AI agent to prioritize attack attempts, we introduced a bias towards the "Send attempt" action in the reward function. Additionally, we forced the agent to perform an attempt action after three other actions, ensuring a more proactive attack strategy.
- **Fast Information Sharing**: To enable rapid information sharing and coordination among multiple AI agents operating in parallel environments, we implemented shared memory for persistent data across threads. This solution facilitated faster learning and adaptation to defensive measures.
- **Environment Reset Delays**: Resetting and switching environments between training iterations caused delays. To mitigate this, we implemented helper queries to reset and switch environments immediately after a query was sent, improving the efficiency of the training process.
- **Each Environment is Different**: The varying characteristics of each testing environment required tailoring the AI agent's strategies for optimal performance. We addressed this by tweaking the reward function to acknowledge the specific challenges and nuances of each environment, promoting adaptability.

These solutions proved crucial in shaping the AI agent's behavior, aligning its learning process more closely with the complexities of real-world brute force attack scenarios.

4 Results and Analysis of AI-Driven Brute Force Attacks

This chapter presents the results obtained from testing our AI agent's performance in various simulated environments. We analyze the effectiveness of the AI-driven brute force attacks, highlighting key findings and observations. The chapter also includes a discussion of the challenges encountered and potential areas for improvement.

4.1 Environment 1: Baseline Testing

Environment 1 (WebApp1/App1.py) serves as a baseline, representing a simple login application with no lockout mechanisms. We test this environment using only middle-ware detection with no blocking. As depicted in Figure 2, our AI agent successfully stabilized its attack rate and blocks, achieving a consistent mean reward. This shows that the agent rapidly understand how to undertake the middle-ware measures. However, the agent was heavily penalized for actions different to sending an attempt. Figure 2 shows that, at the end it could overcome network protection, but still uses unnecessary actions, staying in a stable "negative" reward, that is, without suffering the heavy penalization of being detected. Future work could involve incorporating time taken into the reward function, in addition to the number of attempts, to encourage more focused behavior.

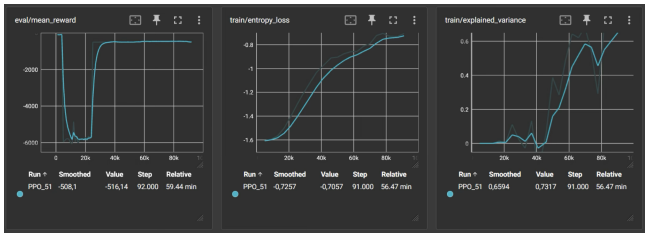

Figure 2: Environment 1

4.2 Environment 2: User-Based Blocking

Environment 2 (WebApp2/App2.py) simulates a more realistic scenario with user-based blocking. This web application enforces a temporary lockout after five failed login attempts within a five-minute window. The global counter used to track failed login attempts ensures consistent enforcement across all sessions. Figures 3 and 4 illustrate our agent's performance in this environment with and without network protection. The consistent penalization of inaction and blocking strategies encouraged our AI agent to explore and adapt its approach. After a considerable number of rounds (20,000 without network protection and 70,000 with network protection), the agent began to demonstrate an understanding of the need to shift strategies periodically to optimize performance.

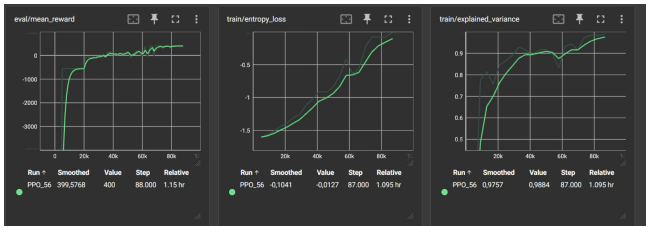

Figure 3: Environment 2 without network protection (20k rounds until optimal strategies begin to emerge)

Figure 4: Environment 2 with network protection (70k rounds until optimal strategies begin to emerge)

4.3 Environment 3: Permanent Blocking Challenges

Environment 3 (WebApp4/App4.py) introduces a significant challenge by implementing permanent account blocking after three failed login attempts. This strict security measure aims to effectively prevent brute-force attacks. Additionally, if more than 50 out of the 100 total users are blocked, the application returns error code 510 to mitigate potential denial-of-service scenarios caused by excessive blocks. As shown in Figure 5, our AI agent struggled to adapt to permanent blocking. This limitation can be attributed to several factors:

- **Limited Action Scope**: The AI agent's primary actions, such as changing IPs, adjusting attack methods, manipulating timing, and skipping users, prove ineffective against permanent blocking. Once an account is locked, it remains inaccessible regardless of the agent's efforts.
- **Reinforcement Learning Limitations**: The reinforcement learning approach, where the agent learns from rewards and penalties, encounters difficulties in a scenario with permanent blocking. Once an account is locked, the AI agent receives no further feedback, hindering its ability to learn and adapt.
- **Inability to Recognize Permanent State Change**: The AI agent may not possess the capacity to fully comprehend the concept of a permanent state change. It may continue to attempt variations of learned strategies without recognizing that the target account is permanently inaccessible.

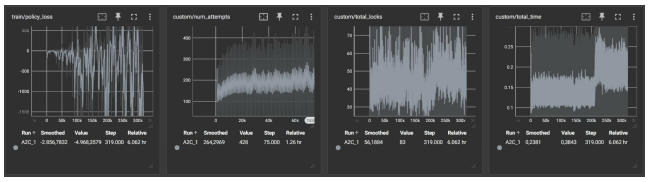

Figure 5: Environment 3

4.4 Environment 4: FTP Server Testing

Environment 4 (IH3A/IH3A/FTP.py) shifts the focus to testing our AI agent's effectiveness against an FTP server. This environment utilizes a vanilla implementation of VSFTPd, relying on standard OS login procedures and local user accounts for authentication. Figures 6 and 7 illustrate the agent's performance with and without network protection. As we can see, the results are similar to our baseline environment.

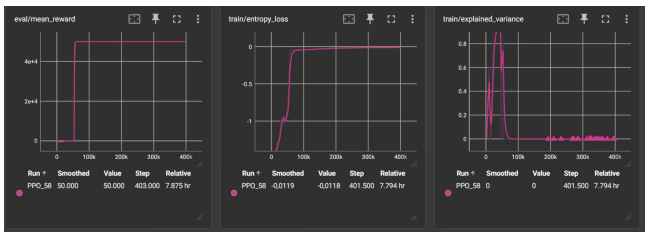

Figure 6: FTP environment with no network protection

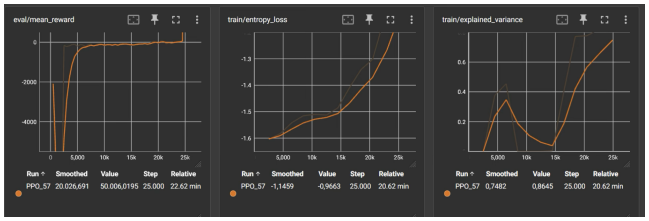

Figure 7: FTP environment with network protection

The findings suggest that network protection did not significantly hinder our AI agent's attack strategies in this environment. However, further research is needed to explore specific rules or configurations for other protocols that might enhance protection.

4.5 Simulated Environment: Streamlining Training with Limitations

To enhance training efficiency, we developed a simulated environment and compared it to Environments 2 and 4 (temporary lockout). The aim was to address challenges associated with real-world training, such as time consumption, resource intensity, and potential disruptions to live systems. The simulated environment aimed to replicate the core logic of authentication and brute force protection mechanisms using a simplified, function-based approach. While the simulated environment demonstrated similar outcomes to real environments in some aspects, it revealed limitations in accurately simulating real-world timing and complexities, particularly those involving advanced security measures. Despite these limitations, well-designed simulations could still prove valuable for AI agent

training in cybersecurity. However, further research is needed to develop more sophisticated simulation techniques or hybrid approaches that combine simulation speed with real-world realism.

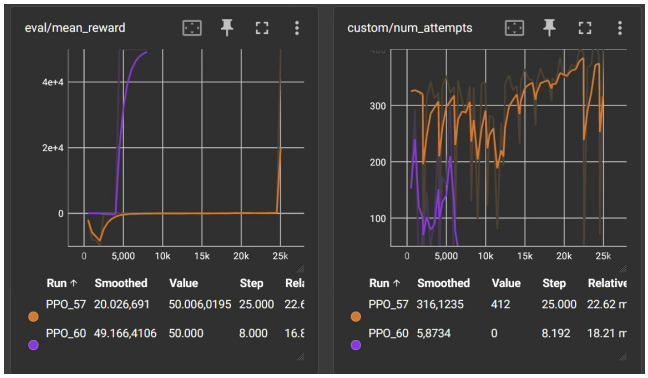

Figure 8: Simulated Environment

4.6 Key Findings and Observations

Our AI agent's performance across the four tested environments, as well as the simulated environment, revealed valuable insights into both the capabilities and limitations of AI-driven attacks against varying security configurations.

One of the most notable findings is the effectiveness of AI agents against temporary blocking mechanisms, such as those implemented in Environment 2. The agent's ability to adjust its attack strategies and timing, incorporating tactics like changing IPs and strategic waiting, enabled it to successfully circumvent these defenses. However, our research also highlighted a significant limitation: permanent blocking mechanisms, as seen in Environment 3, presented a major challenge for the AI agent. This difficulty stemmed from the AI's limited action scope and the inherent nature of reinforcement learning, which relies on feedback to learn and adapt. Once an account was permanently blocked, the AI received no further feedback, preventing it from refining its strategy. Additionally, the AI may not have been able to grasp the concept of a permanent state change, leading it to persist with ineffective tactics.

The experiments also revealed the vulnerability of certain services, particularly FTP servers, to AI-driven brute force attacks. The tests conducted on a vanilla implementation of VSFTPd in Environment 4 demonstrated that the lack of robust anti-brute force mechanisms in many services can make them easy targets for AI-powered attacks, emphasizing the need for comprehensive security across all access points.

Another important observation relates to the complex interplay between middleware solutions and application-level security measures. We found that, in some instances, the presence of an IPS or WAF could inadvertently aid attackers by interfering with application-level anti-brute force mechanisms. This unexpected outcome highlights the importance of carefully coordinated security measures to avoid creating unintended vulnerabilities.

Finally, the development and testing of a simulated environment underscored the potential benefits and ongoing challenges associated with this approach. While simulated environments offer advantages in terms of training efficiency and scalability, accurately simulating real-world timing and the full complexity of advanced security measures remains a difficult task. The simulated environment provided similar outcomes to the real-world environments in certain aspects, but its limitations in replicating timing nuances and intricate interactions highlighted the need for more advanced simulation techniques or hybrid approaches to achieve greater realism.

5 Future Directions: Enhancing AI Agent Capabilities

Building upon our findings and observations, several promising directions for future research emerge:

- **Advanced Reinforcement Learning and LLM's**: Exploring more sophisticated reinforcement learning algorithms, such as Proximal Policy Optimization (PPO), could improve the AI agent's adaptability and performance in complex scenarios. LLM's using COT [11] and Agentic AI Frameworks [10] could derive in a more comprehensive understanding of the environments and could help to solve more efficiently unknown environment.

- **Parallel Environment Processing**: Enhancements to parallel environment processing techniques are crucial for coordinating multiple AI agents efficiently. This would involve addressing synchronization challenges and optimizing information sharing among agents.

- **Expanded Attack Vectors**: Expanding the simulated environments to encompass a wider range of cyberattacks, including SQL injection, cross-site scripting, and advanced persistent threats, would provide a more comprehensive assessment of the AI agent's capabilities.

- **Integration with Attack Tools**: Integrating AI-driven techniques into real-world attack tools could automate attack processes, optimize strategies, and improve overall efficiency.

- **Dynamic Environment Adaptation**: Enabling the AI agent to dynamically adapt to changing environments and unforeseen security measures would enhance its effectiveness and resilience in real-world scenarios.

6 Impact

The research presented in this chapter demonstrates the significant potential of AI in enhancing offensive capabilities, particularly in the context of brute force attacks. The AI agent successfully breached multiple simulated environments, highlighting the need for robust and adaptive security measures. However, the research also revealed limitations in the AI agent's ability to adapt to certain security configurations, particularly those involving permanent blocking. This underscores the importance of continuous innovation in defensive strategies to stay ahead of evolving threats in environments where permanent blocking is not an answer.

The findings and insights from this research contribute to a better understanding of the complex interplay between AI-driven attacks and defensive measures. Future research should focus on addressing the identified challenges and exploring new avenues for both offensive and defensive AI applications in cybersecurity. By striving for a balance between these capabilities, we can foster a more secure and resilient digital landscape.

7 Apendix: Middleware Rules

Mirantis(1) for Mod Security: [5]

```
# Initialize a counter for brute force
    attempts
SecAction "id:100001,phase:1,nolog,
    pass,initcol:ip=%{REMOTE_ADDR},
    initcol:global=%{GLOBAL} \
setvar:ip.brute_force_counter=0,setvar
    :ip.brute_force_time=%{TX.now}"
# Rule to detect failed login attempt
SecRule REQUEST_URI "@rx ^/(login|auth
    )" "phase:2,deny,status:403,id
    :100002,chain"
SecRule RESPONSE_STATUS "@streq 403" "
    setvar:ip.brute_force_counter=+1,
    setvar:ip.brute_force_time=%{TX.now
    }"
# Rule to block IP after 5 failed
    attempts in 1 seconds
SecRule IP:BRUTE_FORCE_COUNTER "@ge 5"
    "phase:2,deny,status:403,id
    :100003,chain"
SecRule TX:NOW "@lt %{IP.
    BRUTE_FORCE_TIME} + 1"
"setvar:ip.brute_force_counter=0,
    setvar:ip.brute_force_time=%{TX.now
    },msg:'Blocking IP for Brute Force
    Attack',log"
```

Custom Rule for Ossec

```
<!-- App login brute force -->
<rule id= 51509 " level="3
    frequency = 5 " timeframe= 1 >
<if_sid>31108</if_sid>
<url_pcre2>login</url_pcre2>
<pcre2>\] "POST \S+login| "POST /
    loginr</pcre2>
<description>WebApp login attempt.</
    description>
</rule>
```

Custom Rule for Suricata

```
alert http any any -> any any (msg:"
    Possible HTTP Brute Force Attack";
    flow:to_server; content:"POST";
    http_method;
content:"password"; http_uri; pcre
    :"/^(login)$/i"; classtype:web-
    application-attack; sid:100001; rev
    :1; threshold:type both, track
by_src, count 5, seconds 1;)
```

References

[1] Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. Pentestgpt: An llm-empowered automatic penetration testing tool. 2023.

[2] Matteo Esposito and Francesco Palagiano. Leveraging large language models for preliminary security risk analysis: A mission-critical case study. 2024.

[3] M. Hadavi and M. Hemmati. Bypassing web application firewalls using deep reinforcement learning. *The ISC International Journal of Information Security (ISeCure)*, 2022.

[4] M. Hemmati and M. A. Hadavi. Using deep reinforcement learning to evade web application firewalls. In *Proceedings of the 2021 18th International ISC Conference on Information Security and Cryptology (ISCISC)*, pages 35–41, September 2021.

[5] Mirantis. Create brute force rules. https://docs.mirantis.com/mcp/q4-18/mcp-security-best-practices/use-cases/brute-force-prevention/create-brute-force-rules.html, 2018.

[6] S. Padakandla, P. K. J., and S. Bhatnagar. Reinforcement learning algorithm for non-stationary environments. *Applied Intelligence*, 50(11):3590–3606, Nov 2020.

[7] M. Shao et al. Nyu ctf dataset: A scalable open-source benchmark dataset for evaluating llms in offensive security. August 2024. Accessed: 2024-10-27.

[8] J. Wang, L. Qixu, W. Di, Y. Dong, and X. Cui. Crafting adversarial examples to bypass flow-&ml-based botnet detectors via rl. In *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 193–204, San Sebastian, Spain, October 2021. ACM.

[9] X. Wang and H. Hu. Evading web application firewalls with reinforcement learning. 2021. Available online at https://openreview.net/pdf?id=m5AntlhJ7Z5.

[10] X. Wang and D. Zhou. Chain-of-thought reasoning without prompting. *arXiv preprint*.

[11] Q. Wu et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint*.