Autonomous Pentest Agent

A Reinforcement-Learning Agent for Web Penetration Testing

Capstone Project Report

A purpose-built vulnerable web application and a reinforcement-learning agent that learns, through trial and error, to discover all three planted vulnerabilities — outperforming a random baseline sixfold. Educational and fully sandboxed.

Arel · RLcapstone.ai

1. Summary

This project pairs a purpose-built vulnerable web application with a **reinforcement-learning agent** that learns — through trial and error, the paradigm behind game-playing AI — to discover all three planted vulnerabilities. Once trained, the agent finds every vulnerability in **4 actions**; a random baseline requires roughly **25** and frequently misses one. The agent outperforms the baseline **sixfold** and generates a structured findings report.

> Educational and sandboxed. All activity targets a purpose-built vulnerable application on localhost. The techniques are standard OWASP Top 10 teaching examples and must only be used on systems you own or are explicitly authorized to test.

2. Motivation

Conventional vulnerability scanners are fast but follow a fixed script and cannot adapt. **Reinforcement learning** can, in principle, learn efficient attack paths, so this project applies it at a scale small enough to understand and explain end to end.

3. The target application

The target is a ~120-line "MiniBank" Flask application on localhost with three classic vulnerabilities planted deliberately:

- SQL injection — a login bypass achievable by injecting into the username field.
- Reflected XSS — the search box reflects and executes injected script.
- Broken access control (IDOR) — altering an identifier in the URL exposes another user's private record.

4. Method

Penetration testing follows a natural sequence — reconnaissance, then exploitation — modeled here as a **Markov decision process**. At each step the agent selects one of twelve actions (crawl the site, attempt a SQL-injection login, attempt an XSS search, request another user's profile, or one of several unproductive actions). Every action is a **real HTTP request** to the live application, and a vulnerability is counted only when the real response confirms it.

The agent learns with **tabular Q-learning** — a 16×12 table of learned action values. It receives **+10** for empirically confirming a new vulnerability, **+2** for performing reconnaissance first, and **−1** per step, so shorter attack paths score higher. The agent is not told which actions are productive; it explores (randomly at first) and updates the table from the rewards it receives, converging over a few hundred episodes.

5. Results

Agent	Actions to find all 3	Found all 3

Trained agent	4	Every run
Random baseline	~25	~50% of runs

Across training, mean episode reward rises from negative values (ineffective exploration that hits the step limit) to approximately **+28**, the reward of the efficient four-step attack: reconnaissance, then one precise action per vulnerability. Learning converts undirected exploration into an efficient, repeatable strategy — roughly **six times faster** than the random baseline.

6. Automated reporting

On completion, the agent generates a structured penetration-testing report: each finding with its severity, OWASP category, impact, and remediation (parameterized queries, output escaping, proper authorization checks). The report is currently template-generated, with a clearly defined integration point where a language model would produce the narrative.

7. Limitations

- A compact sandbox — three known vulnerabilities, twelve actions, one application — sized for understanding, not a production scanner.
- All activity targets a local application built for the purpose, using standard OWASP teaching examples.
- The project is intended for learning and defense: understanding how attacks work in order to prevent them.

8. Future work (v2)

The first defensive component is already published: a detector for AI-driven database ransomware (JADEPUFFER), which pivots the project from offense to defense. Subsequent work includes additional vulnerability classes, a second application to evaluate generalization, and integrating a language model into the reporting and payload-generation stages.