# 南臺科技大學

## 資訊工程系碩士班

## 碩士學位論文

# 基於大型語言模型的網站自動化滲透測試

# Automated Penetration Testing of Websites Based on Large Language Models

研 究 生 ： 謝明軒

指導教授 ： 郭振忠

中華民國一一五年七月

# 南臺科技大學

## 資訊工程系碩士班


## 碩士學位論文

# 基於大型語言模型的網站自動化滲透測試

# Automated Penetration Testing of Websites Based on Large Language Models


研 究 生 ： 謝明軒

指導教授 ： 郭振忠



中華民國一一五年七月

# 南臺科技大學 碩士論文

## 基於大型語言模型的網站自動化滲透測試

研究生：謝明軒

本論文業經審查及口試合格特此證明

論文考試委員

程凱　　楊吳泉

郭振忠

指導教授　郭振忠

所　　長　洪國鈞

中 華 民 國 一 一 五 年 七 月 二 十 一 日

# 摘　　要

隨著網路應用日益普及，網站安全威脅持續升高，滲透測試成爲評估系統安全性的重要手段。近年大型語言模型 (Large Language Model, LLM) 的快速發展爲自動化滲透測試帶來新契機，MAPTA(Multi-Agent Penetration Testing AI)、AWE(Adaptive Agents for Dynamic Web Penetration Testing) 等系統已在 XBOW-104 基準上展現可觀的漏洞捕獲率；然而這些系統均以高階昂貴模型 (GPT-5、Claude Sonnet 4) 爲核心，伴隨顯著的 API(Application Programming Interface) 推論成本。若直接改用成本最低的弱模型 (gpt-5-nano)，實驗顯示弱模型雖能在約六成的題目被導航至正確的漏洞端點，全基準卻僅捕獲約 3% 的旗標。可見其瓶頸不在於發現漏洞，而在於利用漏洞。本研究因此主張，對最便宜的弱模型而言，利用才是比導航更關鍵的瓶頸。

基於此，本研究設計一套導航、偵察、利用三方協作架構：以階層式 Thompson Sampling 搭配四層評分進行導航、以弱模型負責偵察、以端點綁定 (endpoint-bound) 的 Playbook 引擎負責確定性的漏洞利用。在 XBOW-104 基準上以最弱的 gpt-5-nano 評估，純大型語言模型僅捕獲 2.9%，加入 Playbook 後提升至 30.8%，而完整評估的推論成本僅約 4.16 美元，約爲 AWE(7.73 美元) 的 54%、MAPTA(21.38 美元) 的五分之一。據我們所知，本研究首次證明，即使是成本最低的 nano 級弱模型，只要以端點綁定的 Playbook 補足其欠缺的利用能力，也能完成具實用價值的網站自動化滲透。此外，於相同架構下改用能力更強的模型可進一步提升捕獲率，claude-haiku-4-5 達 33.7%、glm-5.2 達 43.3%，顯示端點綁定架構具良好的跨模型泛化性。

**關鍵詞：**大型語言模型、自動化滲透測試、網站安全、低成本、階層式湯普森取樣



南臺科技大學

# Abstract

Penetration testing is an essential means of assessing web application security, and large language models (LLMs) have opened new opportunities for automating it. Systems such as MAPTA and AWE achieve substantial flag-capture rates on the XBOW-104 benchmark, but both rely on high-end models such as GPT-5 and Claude Sonnet 4 that incur significant inference cost. When the cheapest weak models such as gpt-5-nano are substituted, experiments show that although the weak model is navigated to the correct vulnerable endpoint in about 60% of targets, it captures only about 3% of the flags. Its bottleneck therefore lies not in finding vulnerabilities but in exploiting them.

Based on this observation, this research designs a navigation–reconnaissance–exploitation cooperative architecture, in which hierarchical Thompson Sampling with a four-level scorer performs navigation, the weak LLM performs reconnaissance, and an endpoint-bound playbook engine carries out deterministic exploitation. On XBOW-104 with the weakest gpt-5-nano, a pure LLM captures only 2.9%, while adding the playbook raises this to 30.8% at a total inference cost of about US$4.16, roughly 54% that of AWE and one-fifth that of MAPTA. To our knowledge, this is the first demonstration that even a nano-tier weak model can perform practically useful autonomous web penetration testing once an endpoint-bound playbook supplies the exploitation ability it lacks. Stronger models within the same architecture raise the capture rate further, reaching 33.7% with claude-haiku-4-5 and 43.3% with glm-5.2, indicating that the architecture generalizes well across models.

**Keywords:** Large Language Model, Automated Penetration Testing, Web Security, Low Cost, Hierarchical Thompson Sampling



南臺科技大學

# 誌　　謝

兩年的研究所生涯，隨著這篇論文的完成告一段落，這段期間受到許多人的幫助，在此一併致謝。

感謝鄭錦楸老師在研究初期的指導，是老師帶我接觸資訊安全這個領域，也讓我在題目還很模糊的階段，逐步建立起做研究該有的方法與態度，鄭老師退休後，承蒙郭振忠老師接手指導，從研究方向的釐清、實驗的規劃到論文的撰寫，老師都給了許多具體而實際的建議，在我卡關的時候點出問題所在，也給我足夠的空間自行嘗試與修正，兩位老師的指導，是這篇論文得以完成的關鍵。

感謝口試委員程凱教授與楊吳泉教授撥冗審閱本論文，兩位老師在口試時提出許多切中要害的問題與修正建議，涵蓋實驗設計的嚴謹度、數據呈現的一致性與論述的完整性，這些意見讓本研究在改寫後更爲周延，也讓我看見自己在思考上的疏漏。

感謝實驗室的夥伴蔡士煒與陳冠霖，一起打比賽、一起磨練技術的那段日子，是我在資訊安全上成長最快的時期，也感謝實驗室其他夥伴一路上的討論與提醒，讓這段過程不至於孤軍奮戰，謝謝在這段期間持續給我鼓勵的朋友。

最後，謝謝我的家人，感謝你們一直以來的支持與包容，讓我能夠專心完成學業。



南臺科技大學

# 目　　次


**摘　　要** . . . . . **iv**

**Abstract** . . . . . **v**

**誌　　謝** . . . . . **vi**

**目　　次** . . . . . **vii**

**表 目 錄** . . . . . **x**

**圖 目 錄** . . . . . **xi**

**第一章　緒論** . . . . . **1**

1.1 研究背景 . . . . . 1

1.2 研究動機 . . . . . 2

1.3 研究目的 . . . . . 3

1.4 論文架構 . . . . . 4

**第二章　文獻探討** . . . . . **5**

2.1 滲透測試概述與自動化發展 . . . . . 5

2.2 大型語言模型在滲透測試的應用 . . . . . 6

2.2.1 大型語言模型代理與 ReAct 框架 . . . . . 6

2.2.2 大型語言模型驅動的滲透測試系統 . . . . . 6

2.2.3 AutoPT 與滲透測試狀態機 . . . . . 7

2.3 湯普森取樣與多臂吃角子老虎機 . . . . . 8

2.4 知識編碼式攻擊執行與 Playbook . . . . . 8

2.5 XBOW-104 測試基準 . . . . . 9

2.6 相關研究比較與研究缺口 . . . . . 10

**第三章　系統設計與架構** . . . . . **12**

3.1 系統概述與整體流程 . . . . . 12

3.2 搜尋層之階層式湯普森取樣 . . . . . 15

3.2.1 端點表示與 URL 路徑搜尋樹 . . . . . 15

3.2.2 URL 正規化 . . . . . 15

3.2.3 階層式湯普森取樣選擇 . . . . . 16




南臺科技大學

3.2.4 非對稱衰減信用回傳 . . . . . 17
3.2.5 新端點納入 . . . . . 18
3.3 四層評分 . . . . . 18
3.3.1 評分錨點 . . . . . 19
3.3.2 第一層確定性工具解析 . . . . . 19
3.3.3 第二層工作記憶差異與關鍵字比對 . . . . . 19
3.3.4 第三層大型語言模型主張與驗證 . . . . . 20
3.3.5 第四層 OWASP 漏洞類型偵測 . . . . . 20
3.3.6 最終組合 . . . . . 21
3.4 事實庫與 Playbook 引擎 . . . . . 21
3.4.1 事實庫 . . . . . 21
3.4.2 事實萃取器 . . . . . 22
3.4.3 Playbook 結構 . . . . . 23
3.4.4 Playbook 的知識來源與編寫方法 . . . . . 23
3.4.5 Playbook 的數量與類別分布 . . . . . 24
3.4.6 觸發迴圈與控制機制 . . . . . 24
3.4.7 端點綁定 . . . . . 25
3.5 LLM 與工具介面 . . . . . 27
3.5.1 主代理與 ReAct 迴圈 . . . . . 27
3.5.2 分析者代理 . . . . . 27
3.5.3 端點鎖定與去重快取 . . . . . 27
3.5.4 工具介面 . . . . . 28
3.6 工作記憶 . . . . . 28
3.7 停止條件 . . . . . 29
3.8 系統參數與本章小結 . . . . . 29
**第四章 實驗設計與結果 . . . . . 31**
4.1 實驗設置 . . . . . 31
4.2 評估指標 . . . . . 32
4.3 漏洞端點深度分布 . . . . . 32
4.4 推論成本 . . . . . 33
4.5 與既有方法的整體比較 . . . . . 33
4.6 整體捕獲率顯示利用才是瓶頸 . . . . . 34



南臺科技大學

4.7 攻擊類型成功率與 Playbook 消融 . . . . . 35
4.8 導航分析與搜尋演算法的角色 . . . . . 37
4.9 跨弱模型比較 . . . . . 40
4.10 更強模型的表現與展望 . . . . . 40
4.11 捕獲機制剖析 . . . . . 43
4.12 純確定性基準：移除大型語言模型 . . . . . 44
4.13 實際案例一：成功利用之 URL 路徑搜尋樹 . . . . . 46
4.14 實際案例二：失敗案例之 URL 路徑搜尋樹 . . . . . 46
4.15 綜合討論 . . . . . 48
**第五章 結論與未來展望 . . . . . 49**
5.1 研究結論 . . . . . 49
5.2 研究貢獻 . . . . . 50
5.3 研究限制 . . . . . 50
5.4 未來展望 . . . . . 51
**參考文獻 . . . . . 53**
**公式彙整 . . . . . 57**
**附錄 A 系統提示詞全文 . . . . . 59**
A.1 主代理系統提示詞全文 . . . . . 59
A.2 分析者代理系統提示詞全文 . . . . . 61



南臺科技大學

# 表 目 錄

表 2.1 代表性大型語言模型滲透測試系統比較 . . . . . 10

表 3.1 評分錨點及其語意 . . . . . 19
表 3.2 curl 探測之 HTTP 狀態碼至 L1 分數對應 . . . . . 20
表 3.3 第二層工作記憶差異評分之優先順序 . . . . . 20
表 3.4 事實萃取器類別 . . . . . 22
表 3.5 Playbook 的知識來源與涵蓋範圍 . . . . . 24
表 3.6 五十本 Playbook 之漏洞類別分布 . . . . . 25
表 3.7 二十個類型化工具 . . . . . 28
表 3.8 工作記憶之六個區段 . . . . . 29
表 3.9 系統主要參數 . . . . . 30

表 4.1 XBOW-104 漏洞端點之路徑深度分布 . . . . . 33
表 4.2 本研究與既有方法之整體比較 . . . . . 34
表 4.3 各設定之整體捕獲率 . . . . . 34
表 4.4 主組態之捕獲率依漏洞深度分布 . . . . . 35
表 4.5 各攻擊類型之捕獲率與 Playbook 消融 . . . . . 36
表 4.6 純 LLM 三組於各漏洞深度之到位率 . . . . . 37
表 4.7 純 LLM 三組之導航到位率與選擇效率 . . . . . 38
表 4.8 不同演算法與樹大小之純導航比較 . . . . . 39
表 4.9 兩個弱模型於主組態之對照 . . . . . 40
表 4.10 四個模型於主組態之捕獲率與 token 及成本 . . . . . 41
表 4.11 各漏洞類型之跨系統與跨模型捕獲率 . . . . . 42
表 4.12 主組態 32 個捕獲之機制剖析 . . . . . 43
表 4.13 各組態之元件組成與捕獲對照 . . . . . 45
表 4.14 主組態 104 次執行之停止原因分布 . . . . . 45



南臺科技大學

# 圖 目 錄

圖 3.1 系統架構圖 . . . . . 14
圖 3.2 URL 路徑搜尋樹 . . . . . 16

圖 4.1 純 LLM 的導航與利用落差 . . . . . 35
圖 4.2 成功案例之 URL 路徑搜尋樹與 TS 選擇歷程 . . . . . 46
圖 4.3 失敗案例之 URL 路徑搜尋樹與 TS 選擇歷程 . . . . . 47



南臺科技大學

# 第一章

# 緒論

## 1.1 研究背景

隨著數位轉型的加速推進，網路應用已成爲現代企業營運與社會運作的關鍵基礎設施。從電子商務平台、金融交易系統到政府公共服務入口，各類網站承載著大量敏感資料與關鍵業務流程。然而，網路應用的快速發展也伴隨著日益嚴峻的安全威脅。根據 OWASP(Open Worldwide Application Security Project) 持續更新的十大網站安全風險報告 [1]，注入攻擊、身分驗證失效和敏感資料洩漏等漏洞類型長期佔據威脅排行榜前列，對企業與使用者構成重大風險。

滲透測試 (Penetration Testing) 作爲一種主動式安全評估方法，透過模擬眞實攻擊者的視角與手法，系統性地發掘目標系統中的安全漏洞，目前已成爲組織確保資訊安全的重要手段。傳統的滲透測試流程通常包含六個階段：規劃與偵察、掃描與列舉、漏洞利用、後滲透活動、報告撰寫、修復複測。然而，此流程高度依賴專業滲透測試人員的經驗與技術，不僅耗時且成本高昂，更受限於人力資源的稀缺性。隨著網路應用規模的持續擴大，傳統手動滲透測試已難以滿足日益增長的安全檢測需求。

近年來，大型語言模型的快速發展爲這個難題帶來了新的方向。GPT-4、Claude、Gemini 等閉源模型 [2], [3], [4]，以及 DeepSeek、Qwen 等開源模型 [5], [6]，都展現出強大的自然語言理解、程式碼生成與邏輯推理能力，在理解漏洞描述、生成攻擊指令、分析工具輸出與制定攻擊策略上都有明顯的潛力。研究者把大型語言模型和代理 (Agent) 架構結合後，已經初步做到讓智慧代理自動驅動滲透測試的各個階段，不需要人工介入就能完成從偵察、掃描到漏洞利用的完整流程 [7], [8]。不過，現有系統在低成本部署、模型選擇彈性與決策效率上仍有不少改進空間，距離眞正能規模化的自動化滲透測試還有一段距離 [9], [10]。

在這個背景下，近年具代表性的研究包括 MAPTA(2025)[11] 與 AWE(NDSS 2026)[12] 兩套系統。MAPTA 以 GPT-5 作爲核心推理引擎，在業界廣泛採用的



南臺科技大學

XBOW-104 基準 [13] 上達到 76.9% 的旗標捕獲率，是目前已發表文獻中名列前茅的水準；該基準包含 104 個 docker 化的網站滲透挑戰，涵蓋 XSS(Cross-Site Scripting)、SQL 注入 (SQL Injection)、IDOR(Insecure Direct Object Reference)、SSTI(Server-Side Template Injection)、LFI(Local File Inclusion)、認證繞過與多項 CVE(Common Vulnerabilities and Exposures) 弱點。AWE 則採用 Claude Sonnet 4 並結合適應性決策機制，在同一個基準上達到 51.9% 的捕獲率。這兩套系統都證明了大型語言模型用於自動化滲透測試的可行性，也建立了目前這個領域的效能參考基準。然而，它們的高效能同時反映出現階段的一項共同侷限：優異的捕獲率是建立在高階、昂貴的閉源模型之上，與低成本、可規模化的實用需求之間仍有明顯落差。

## 1.2 研究動機

儘管基於大型語言模型的自動化滲透測試已在 MAPTA、AWE 等系統上展現顯著成果，證明了讓智慧代理自動完成從偵察、掃描到漏洞利用的整個流程確實可行，但這些成果大多是在資源充裕的條件下、以單次評估的形式取得的。要從這樣亮眼的基準成績，走到能讓中小型團隊在日常開發中反覆使用的實用工具，中間仍橫著一段不小的距離；而橫在這段距離上、最現實的一道障礙，就是成本。

這道成本障礙的根源，在於這些系統都以高階模型爲核心。如前節所述，MAPTA 以 GPT-5 爲核心，AWE 採用 Claude Sonnet 4，皆屬高階模型。這類高階模型雖具備強大的推理能力，但單位 token 成本偏高。雖然單次基準評估的開銷看似有限，例如 MAPTA 在 XBOW-104[13] 上的總成本爲 21.38 美元、AWE 爲 7.73 美元，但實際研究與部署往往涉及多輪迭代測試、不同目標環境的驗證、消融實驗與超參數調整，累計成本會迅速放大。對於中小型資安團隊、學術研究機構與資安教育環境而言，這樣的成本結構不僅形成採用門檻，也使系統難以支援高頻率的自動化檢測。要在維持捕獲率的同時大幅壓低推論成本，最自然的途徑就是改用單位成本最低的弱模型。

然而，把高階模型直接換成便宜的弱模型 (gpt-5-nano、gpt-5.4-nano) 並沒有那麼簡單，實際觀察顯示這種替換往往造成效能明顯下降 [8], [14], [15]。原因在於現有系統不論採用 ReAct[16]、多代理或狀態機等編排，漏洞利用最關鍵的一步，即構造攻擊 payload 並判讀回應，多半仍由大型語言模型即時完成，因而高度仰



南臺科技大學

賴模型的多步推理能力與穩定的決策品質。值得注意的是，弱模型的失效並非平均分布於各環節：在搜尋與偵察機制的運作下，它通常仍能抵達正確的漏洞端點，眞正卡關的是站上端點後的利用這一步。本研究第四章將直接量化此一落差：在多數已到達漏洞端點的題目上，弱模型最終完成利用的比例極低。可見瓶頸不在導航而在利用，問題不在於弱模型能力不足，而在於現有架構要求它自行承擔最不擅長的攻擊構造。

弱模型在利用上的困境，其實映照出一個更深層、所有系統都面臨的結構問題：攻擊執行與語意推理被綁得太緊、全部混在一起。現有研究多半把漏洞判定、payload 構造與結果驗證這三件事全部交給大型語言模型在同一條推理鏈中完成 [7], [8], [17]。然而這三件事多半具有高度結構化、可重複的特性：payload 構造有 XSS 標籤變體、SQL 注入語法、SSTI 範本語法等固定的語法模式，漏洞類別的判定可以由錯誤訊息、回應標頭、特定範本例外等工具輸出中的特徵字串推斷，結果驗證也能以旗標樣式、命令執行回顯等明確的成功條件來認定 [9], [10], [18]。把這些原本可以用確定性規則處理的工作交給大型語言模型，不僅把昂貴的推論 token 浪費在可以事先編碼的步驟上，也讓系統的行爲受制於模型推理的不穩定性。近期研究也支持這個觀察：AWE 以較弱的 Claude Sonnet 4 模型，透過把領域知識編碼進專化的攻擊流程，在注入類漏洞上反而勝過使用 GPT-5 的 MAPTA。這顯示若能把確定性執行與語意決策適度分開，讓大型語言模型專注在少量但關鍵的高層次判讀，便有機會在大幅降低成本的同時維持甚至提升整體效能。

綜合上述三項觀察，本論文形成一項核心主張：在自動化滲透測試系統中，大型語言模型的主要價值在於偵察與協調，而非獨自承擔最不擅長的攻擊構造；高度結構化、可重複的漏洞利用，則由確定性的執行模組補足，與大型語言模型共同承擔。唯有把弱模型最不擅長的結構化利用交由確定性模組補足，才可能在把核心模型換成成本最低的弱模型時，仍維持具實用價值的效能。

## 1.3 研究目的

基於前述研究動機，本研究聚焦於一個尚未被充分探討的問題：對最低成本的弱模型而言，網站自動化滲透的眞正瓶頸是什麼。爲此，本研究設計並實作一套將導航與利用明確分開的自動化滲透系統，並以系統化實驗分離量測兩種能力。據此，本研究提出三個研究問題：



南臺科技大學

**RQ1**：對成本最低的弱模型而言，利用是否爲比導航更關鍵的瓶頸？

**RQ2**：端點綁定的 Playbook 能否補足弱模型的利用缺口，使捕獲率達到具實用價值的水準？

**RQ3**：導航是否構成弱模型的瓶頸？湯普森取樣 (Thompson Sampling, TS)[19]、置信上限演算法 (Upper Confidence Bound, UCB1)[20] 與隨機選擇 (Random)[21] 三種搜尋演算法，在把弱模型帶到漏洞端點的導航能力上是否有顯著差異？

具體而言，本研究方式有四個。第一是設計一套將導航與利用明確分開的低成本自動化滲透架構，以階層式 Thompson Sampling 搭配四層評分負責導航。第二是提出端點綁定的 Playbook 引擎，與大型語言模型共同承擔漏洞利用，補足弱模型自行構造結構化攻擊的不足。第三是在 XBOW-104 基準上以最低成本的 gpt-5-nano 驗證系統可行性，並透過分離量測導航與利用釐清弱模型的眞正瓶頸。第四是與 MAPTA、AWE 等採用高階模型的系統比較推論成本，凸顯本系統的成本效益。

## 1.4 論文架構

本論文共分爲五章，第一章說明研究背景、動機、目的與論文架構，第二章爲文獻探討，回顧滲透測試自動化的發展、大型語言模型在滲透測試的應用、湯普森取樣與多臂吃角子老虎機、知識編碼式攻擊執行，以及 XBOW-104 測試基準，並歸納研究缺口。第三章說明本研究提出的系統設計與架構，依序介紹負責導航的階層式湯普森取樣與四層評分、事實庫、負責利用的端點綁定 Playbook 引擎、ReAct 代理與類型化工具、工作記憶與停止條件。第四章說明實驗設計與結果：先以實驗設置、評估指標與漏洞端點深度分布鋪陳，接著集中呈現各項比較，包含推論成本、與既有方法就架構、捕獲率與成本的整體比較、Playbook 消融、攻擊類型的消融、搜尋演算法比較，以及跨不同模型的比較。最後以捕獲機制剖析、移除大型語言模型的純確定性基準與案例研究深入分析，並歸納綜合討論。第五章爲結論與未來展望。



南臺科技大學

# 第二章

# 文獻探討

滲透測試要眞正自動化，關鍵在於讓系統能在無人介入下完成從偵察、掃描到漏洞利用的完整流程。大型語言模型的出現使這一目標在近年快速逼近，卻也帶來一個現實的張力：現有系統可觀的捕獲率多半建立在高階、昂貴的模型之上，與低成本、可規模化的實務需求之間存在明顯落差。由此浮現一個尚未被充分回答的問題：對成本最低的弱模型而言，自動化滲透是否仍可行、其眞正的瓶頸又在何處？本章透過回顧相關文獻，逐一檢視這個問題的各個面向，並據以論證現有研究在低成本弱模型上仍存在明顯的缺口。

## 2.1 滲透測試概述與自動化發展

滲透測試是一種透過模擬眞實攻擊行爲來評估資訊系統安全性的防禦技術，其標準流程通常涵蓋規劃與偵察、掃描與列舉、漏洞利用、後滲透活動、撰寫報告及修復複測等階段。這些階段要求測試人員具備廣泛的安全知識，包括對各類漏洞的理解、工具的熟練操作，以及不同階段間資訊的有效整合。長期以來，滲透測試高度依賴人工操作與專家經驗，自動化程度有限 [22]。

在自動化滲透測試的發展歷程中，早期研究主要探索基於規則 (Rule-based) 與深度強化學習 (Deep Reinforcement Learning) 的方法。基於規則的方法透過預定義的決策樹或規則集來指導測試流程，但其適應性受限於規則覆蓋的範圍；強化學習方法如 DynPen[23] 則嘗試在動態網路環境中訓練自動化滲透測試代理，雖可應對環境變化，但需要大量訓練資料且難以泛化至未見過的環境。上述方法雖在特定場景取得進展，但均無法實現眞正的全自動 Web 滲透測試，即在無人介入的情況下完成整個測試流程並適應多樣化的環境。

近年來，大型語言模型的快速發展爲滲透測試自動化帶來了新的契機。大型語言模型展現出強大的推理與泛化能力，使其能夠作爲智慧代理參與滲透測試的各個階段 [24]。相較於傳統方法，大型語言模型代理能夠理解自然語言描述的任



南臺科技大學

務目標、動態生成測試指令，並根據環境回饋調整策略，大幅提升了自動化的可行性與靈活度。

## 2.2 大型語言模型在滲透測試的應用

### 2.2.1 大型語言模型代理與 ReAct 框架

大型語言模型代理 (LLM Agent) 是指以大型語言模型爲核心、結合外部工具使用能力的人工智慧實體，能感知環境、做出決策並採取行動。Xi 等人 [24] 在其綜述中系統性地回顧了大型語言模型代理的發展，提出以大腦、感知、行動三大組件爲核心的通用框架，並指出大型語言模型憑藉其多功能性被視爲通用人工智慧的潛在基石。

在代理推理框架方面，ReAct[16] 是目前最具代表性的方法之一。ReAct 將推理 (Reasoning) 與行動 (Acting) 進行協同，使大型語言模型在每一步中先進行思考 (Thought)，再決定採取的行動 (Action)，最後觀察環境回饋 (Observation)，形成迭代式的推理-行動迴圈。此框架已被廣泛應用於需要與環境互動的任務中，包括程式碼執行、網頁操作及滲透測試等場景。然而，ReAct 在面對長期任務時存在上下文溢出與推理偏移的問題 [16]，容易導致代理陷入局部迴圈而無法自拔。值得注意的是，ReAct 的決策品質高度依賴模型本身的推理能力；當核心模型改用弱模型時，過高的決策自由度反而成爲負擔，這正是本研究試圖以確定性執行模組加以緩解的問題。

### 2.2.2 大型語言模型驅動的滲透測試系統

隨著大型語言模型能力的提升，多個基於大型語言模型的滲透測試框架相繼被提出。PentestGPT[17] 是較早期的代表性工作，由 Deng 等人於 USENIX Security 2024 發表，採用 ReAct 風格的代理架構並結合滲透測試任務樹 (Penetration Testing Task Tree) 來輔助半自動化測試。PentestGPT 包含解析、推理與生成三個模組，但其本質上是一個人機協作 (Human-in-the-Loop) 系統，仍需滲透測試人員作爲執行代理進行操作，且缺乏標準化的基準測試環境與可機器驗證的成功標準。

PENTEST-AI[25] 提出基於 MITRE ATT&CK 框架的多代理系統，將滲透測試的不同階段分配給專門的代理，強調代理間的協作與分工；PenHeal[26] 則提出



南臺科技大學

兩階段框架，先進行漏洞偵測再進行自動修補。然而這些研究多未提供標準化的測試環境或自動化的成功驗證機制，影響了實驗結果的可重複性。

最新的研究 MAPTA[11] 由 David 與 Gervais 於 2025 年提出，是目前已知的首個開源多代理 Web 滲透測試系統。MAPTA 採用協調者 (Coordinator)、沙箱代理 (Sandbox Agent) 與驗證代理 (Validation Agent) 三角色架構，結合工具導向的執行與端對端的漏洞利用驗證。在 XBOW-104 中，MAPTA 達到 76.9% 的整體捕獲率，其中 SSRF 與設定錯誤類漏洞達 100% 捕獲率。該研究展示了多代理協作的潛力，但其架構複雜度較高，且主要依賴單一商用模型 GPT-5，伴隨高昂的推論成本。

與 MAPTA 同期，Jaswal 與 Baghel 提出的 AWE[12] 則走向另一條路徑。AWE 採用記憶增強的多代理框架，將漏洞類別專屬的分析流程嵌入輕量級的大型語言模型編排層，並以瀏覽器後端驗證確保結果的確定性與可重現性。值得注意的是，AWE 僅使用中階的 Claude Sonnet 4 模型，於 XBOW-104 上達到 51.9% 的整體捕獲率，並在部分注入類漏洞上以更低的成本與 token 用量超越使用 GPT-5 的 MAPTA。AWE 的結果揭示了一個重要訊息：將領域知識編碼進專化的攻擊流程，可能比單純提升模型能力更具成本效益，這正是本研究架構設計的重要啓發。

### 2.2.3 AutoPT 與滲透測試狀態機

AutoPT[22] 由 Wu 等人發表於 IEEE Transactions on Information Forensics and Security(TIFS, 2025)，是一個基於滲透測試狀態機 (Penetration Testing State Machine, PSM) 的全自動 Web 滲透測試系統。其核心創新在於將有限狀態機 (Finite State Machine)[27] 的概念引入大型語言模型驅動的滲透測試流程，透過結構化的狀態轉換來管理代理的決策過程。AutoPT 將任務分解爲掃描、偵察、漏洞利用等代理狀態，與漏洞選擇、完成檢查等規則狀態，並以 GPT-4o mini 等模型驅動，整體任務完成率較 ReAct 基線有顯著提升，且其模組化分解設計能有效降低對特定商用模型的依賴。AutoPT 以工具硬編碼綁定於各狀態，反映了將領域知識結構化的設計趨勢，但其工具與狀態綁得太死，也限制了系統的擴充彈性。



南臺科技大學

## 2.3 湯普森取樣與多臂吃角子老虎機

湯普森取樣源於多臂吃角子老虎機 (Multi-Armed Bandit) 問題：決策者面對多個報酬分布未知的選項，須在有限次嘗試中最大化累積報酬，其核心是在探索與利用之間權衡，前者是嘗試不確定的選項以獲取資訊，後者是選擇已知表現較佳的選項。湯普森取樣最早由 Thompson 於 1933 年提出 [19]，是處理這種權衡最古老的啓發式之一。其核心思想是：對每個可選行動維護一個關於其報酬機率的後驗分布，每次決策時從各行動的後驗中各抽樣一次，並選擇抽樣值最大的行動。如此一來，後驗不確定性較高的行動仍有機會被選中，形成探索；而歷史表現較佳的行動則有較高機率被選中，形成利用，兩者在機率意義下自然取得平衡。

在成功或失敗的二元報酬情境下，湯普森取樣常以 Beta-Bernoulli 共軛模型實作：每個行動維護一組 Beta($\alpha$, $\beta$) 參數，初始爲均勻先驗 Beta(1, 1)；每次觀察到成功則 $\alpha$ 增加、失敗則 $\beta$ 增加，其報酬機率 $\theta$ 的後驗期望值爲

$$\mathbb{E}[\theta] = \frac{\alpha}{\alpha+\beta}. \tag{2.1}$$

由於 Beta 分布爲 Bernoulli 分布的共軛先驗 [28]，後驗更新爲封閉形式、計算成本低，特別適合線上決策。Chapelle 與 Li[29] 在大規模實證研究中指出，湯普森取樣在展示廣告與新聞推薦等眞實任務上的表現可媲美甚至優於 UCB 等主流方法，且對延遲回饋更爲穩健；Russo 等人 [30] 則提供了湯普森取樣的系統性教學與後悔界 (regret bound) 分析，進一步奠定其理論基礎。

傳統湯普森取樣多應用於扁平的行動集合，但網站滲透測試的端點依 URL 路徑具有天然的階層結構。將賭博機制延伸至樹狀結構的概念可追溯至 Kocsis 與 Szepesvári 提出的 UCT(Upper Confidence bounds applied to Trees)[31]，其在蒙地卡羅樹搜尋中以賭博演算法引導樹的擴展，成功應用於賽局決策。本研究借鑑其在樹上進行賭博式搜尋的思路，將湯普森取樣延伸至 URL 路徑搜尋樹上的階層式搜尋，其具體機制詳見第三章。

## 2.4 知識編碼式攻擊執行與 Playbook

在實務滲透測試中，許多漏洞的利用步驟是高度結構化且可重複的，這類可重複的攻擊流程可整理爲 Playbook。例如跨站腳本 (XSS) 的各種標籤變體、SQL



南臺科技大學

注入的 UNION 與布林盲注語法、伺服器端模板注入 (SSTI) 的範本語法、路徑遍歷的 `php://filter` 鏈，甚至特定 CVE 的利用序列，都可以事先用參數化的模板編碼。資安社群長期累積的知識庫，例如 OWASP 測試指南、HackTricks、PayloadsAllTheThings 等，本質上即是此類 Playbook 的知識來源。

將這類知識編碼爲確定性的執行模組，相較於每次都由大型語言模型即時生成 payload，具有三項優勢：其一，避免昂貴的推論 token 浪費在原本可預先編碼的結構化步驟上；其二，確定性執行不受模型幻覺影響，可重現且可稽核；其三，可在不啓用大型語言模型的情況下獨立運作。AWE[12] 將漏洞類別專屬的分析流程嵌入編排層、AutoPT[22] 以狀態機綁定工具，都反映了將領域知識結構化的設計趨勢。然而，這些系統大多把腳本與特定狀態或代理硬性綁定，缺乏一個以可觀察事實爲觸發條件、能在搜尋過程中動態決定何時觸發何種腳本的通用機制。本研究提出的端點綁定 Playbook 引擎正是針對這個缺口而設計，其以可觀察事實爲觸發條件、並將觸發限制在已發現的端點上，具體機制詳見第三章。

## 2.5 XBOW-104 測試基準

XBOW-104[13] 是由 XBOW 安全團隊委託外部承包商開發的一套 Web 應用程式安全測試基準，旨在爲自動化滲透測試框架提供公平且可重複的評估環境。該基準由 104 個獨立的安全挑戰組成，採用搶旗 (Capture The Flag, CTF) 形式設計，每個挑戰對應一種眞實世界中常見的 Web 漏洞類型。

XBOW-104 具備以下重要特點。首先，在防記憶與可重現方面，每個挑戰封裝於獨立的 Docker 容器中，其旗標於每次建置時以隨機值重新生成並注入，使每次測試的旗標值都不同；因此即使題庫已公開於程式碼託管平台、挑戰內容可能已被納入模型的訓練語料，代理仍無法憑記憶取得旗標，必須實際利用漏洞才能捕獲，這確保了評估的公平性與客觀性。在漏洞類型覆蓋方面，XBOW-104 涵蓋了多種 OWASP Top 10[1] 風險類別，包括 SQL 注入、命令注入、伺服器端模板注入等注入攻擊，以及跨站腳本、伺服器端請求僞造、權限控制失效、安全設定錯誤等。MAPTA[11] 的評估結果顯示，不同類型漏洞的自動化利用難度存在顯著差異，其中 SSRF 與設定錯誤類較易自動化，而跨站腳本與盲注入則對自動化代理構成較大挑戰。

相較於 AutoPT[22] 原有的 20 個 CVE 基準環境，XBOW-104 在規模 (104 對 20)、漏洞多樣性以及動態旗標注入的防作弊機制等方面都有明顯優勢，能爲評估



南臺科技大學

自動化滲透測試框架提供更嚴謹的測試環境，因此本研究選用它作爲主要評估基準。

## 2.6 相關研究比較與研究缺口

綜合上述文獻回顧，表 2.1 彙整了代表性系統在核心模型、搜尋/決策策略、知識編碼方式與捕獲率上的差異。需說明的是，PentestGPT 屬人機協作、未提供標準化的自動化捕獲率，AutoPT 則於另一套 20 題 CVE 基準評估，兩者均未在 XBOW-104 基準上測試，其捕獲率無法與 MAPTA、AWE 直接對照，故表中以「—」表示；MAPTA 與 AWE 則同在 XBOW-104 上評估，可直接比較。現有大型語言模型驅動的滲透測試研究在以下幾個面向仍有明顯的缺口。

**表 2.1** 代表性大型語言模型滲透測試系統比較

| 系統 | 核心模型 | 搜尋/決策策略 | 知識編碼 | 捕獲率 |
|---|---|---|---|---|
| PentestGPT[17] | GPT-4 | ReAct + 任務樹 (人機協作) | 無 | — |
| AutoPT[22] | GPT-4o mini 等 | 滲透測試狀態機 (PSM) | 工具綁定於狀態 | — |
| MAPTA[11] | GPT-5 | 多代理自由探索 | 驗證代理 | 76.9% |
| AWE[12] | Claude Sonnet 4 | 輕量編排＋記憶 | 類別專屬流程 | 51.9% |

第一，現有研究都依賴高階昂貴模型，沒有探討最低成本弱模型的可行性。MAPTA 採用 GPT-5、AWE 採用 Claude Sonnet 4、PentestGPT 系列以 GPT-4 爲主，都屬於高階模型。少數探討較小模型的研究 [14]，則以 8B 級開源模型評估 Linux 主機的提權任務，指出小型模型難以獨力完成自主滲透、但良好的架構設計可部分彌補其能力不足；惟該研究針對的是主機層的提權任務，與本研究關注的網站滲透場景不同。至今尚無研究探討 nano 級的弱模型是否能勝任網站自動化滲透，這類模型的單位成本最低，但推理深度有限，在現行以 ReAct 自由探索爲主的架構下幾乎無法獨力構造攻擊；能否透過適當的架構設計使其可行，是一個尚未被回答的重要問題。

第二，高階模型的推論成本構成採用門檻。MAPTA 與 AWE 雖達到可觀的捕獲率，但其成本結構，單次完整評估需 7.73 至 21.38 美元，在多輪迭代、持續整合與持續部署的反覆評估等實際情境下會迅速放大，對中小型資安團隊與資安教育環境形成門檻。

第三，執行與語意推理綁得太緊。多數框架將漏洞判定、payload 構造與結果驗證全部交由同一大型語言模型在單一推理鏈中完成，既浪費推論成本，也使



南臺科技大學

語意判讀能力被冗餘步驟稀釋，缺乏一種把確定性執行與語意決策適度分開的架構。

第四，搜尋預算分配缺乏原則性方法。在端點數量龐大而漏洞稀疏的 Web 目標上，如何有原則地將有限的測試預算分配給最可能藏有漏洞的端點，現有以 ReAct 自由探索爲主的方法缺乏明確機制，容易在無關端點上耗盡預算。



南臺科技大學

# 第三章

# 系統設計與架構

本研究提出的網站自動化滲透測試系統採三方協作設計，把導航、偵察、利用三件事拆開，各自獨立、分工處理：

- **導航**：在 URL 路徑搜尋樹 (以下簡稱搜尋樹) 上進行階層式 Thompson Sampling，搭配融合工具解析、工作記憶差異、模型主張驗證與 OWASP 漏洞類型偵測的四層評分，產生穩健的獎勵訊號，使有限的測試回合優先集中在最可能藏有漏洞的端點上。
- **偵察**：以 ReAct 迴圈讓便宜的弱模型在被導航到的端點上觀察表單、輸入、框架等特徵，並蒐集爲可觀察事實。
- **利用**：以 Playbook 引擎負責確定性的漏洞利用。

Playbook 並非走到漏洞點便直接套用預存答案的盲目執行，而是先由偵察判讀端點所暴露的工具訊號落入認證繞過、資源列舉、注入等哪一類漏洞，再據此選取並觸發該漏洞類別對應的 Playbook，對該端點發動與其特徵相符的針對性攻擊。換言之，觸發的條件是實際到達、且觀察事實符合特定漏洞類別條件的端點，而實際施加的攻擊內容則由所判定的漏洞類別決定，故稱爲端點綁定。

## 3.1 系統概述與整體流程

給定一個目標網址後，系統自動探索並利用 HTTP 層的安全漏洞，若成功則回報所擷取的旗標。系統的整體架構如圖 3.1 所示，以回合式的學習迴圈運作：每一回合選定一個端點進行滲透、並依該回合結果更新對各端點的信念，逐回合把測試集中到更可能藏有漏洞的路徑。其中大型語言模型可自由替換，本研究刻意採用單位成本最低的弱模型。

流程始於 Bootstrap 探測：系統先以 whatweb 對目標做一次技術指紋辨識，將偵測到的框架與伺服器資訊寫入工作記憶；接著以 dirsearch 搭配其預設約近



南臺科技大學

萬條目的內建字典 dicc.txt，對目標做一次遞迴深度 2 的淺層目錄列舉，作爲端點發現的主要來源，並將列舉到的入口目錄納入初始搜尋樹。dirsearch 亦保留爲 ReAct 代理在主迴圈中可自行呼叫的工具。發現的端點再依 URL 路徑前綴組織成搜尋樹；樹中每個節點以 Beta($\alpha$, $\beta$) 後驗表示該端點可被利用的信念。主迴圈的核心是階層式 Thompson Sampling(HierTS) 選擇器：它先廣掃各端點群、再深入信念較高的子樹，依各節點的 Beta 後驗抽樣選出本回合待測的端點。選定端點後，由 LLM 導航器依事實庫中已累積的觀察事實，選擇要在該端點上使用的工具，交由工具執行模組實際執行並產生輸出。

工具的輸出具有雙重用途：其一，經四層評分產生該回合的獎勵，回寫並更新對應端點節點的 Beta 後驗、並採非對稱衰減回傳，形成回合學習迴圈，使搜尋逐步集中於信念較高的路徑；其二，經事實萃取器解析爲結構化的觀察事實，如觀察到的表單、輸入點、漏洞類別指標等，寫入事實庫。此外，無論本回合是廣掃或深入，每完成一次端點的滲透工作階段，即由弱模型將該回合觀察到的結果摘要並更新至一份跨回合累積的工作記憶文件，該文件分爲已發現端點、關鍵發現、已嘗試攻擊等六個區段 (見第 3.6 節)；此文件既作爲下一回合的提示上下文以避免重複推導，也作爲四層評分第二層偵測進展的輸入。

Playbook 引擎則以事實庫爲觸發條件：當某端點累積的事實符合特定漏洞類別的條件時，對應的 Playbook 便在該端點上自動執行一連串確定性的利用步驟以嘗試取得旗標，成功則交由判斷結束模組輸出，失敗則將結果寫回事實庫供後續回合參考。最後，判斷結束模組檢查終止條件：若已取得旗標則結束並輸出，否則進入下一回合，由 HierTS 重新抽樣。當旗標被擷取，或回合數與時間預算耗盡時，整個迴圈終止。

本系統的回合式學習迴圈整體如演算法 1 所示：每一回合先以階層式湯普森取樣選出端點 (第 3 行，選擇程序見演算法 2)，由弱模型於該端點執行工具，再以四層評分 (第 5 行，詳見後節) 將工具輸出轉爲回合獎勵 $s$，據以更新該端點的 Beta 後驗並沿父節點鏈非對稱回傳，最後由事實庫觸發端點綁定 Playbook 嘗試取得旗標。

演算法中各符號意義如下：$T$ 爲搜尋樹、$v$ 爲本回合選中的端點、$O$ 爲在該端點執行工具所得的輸出；$s \in [0, 0.95]$ 爲四層評分 ($L_1$ 工具解析、$L_2$ 工作記憶差異、$L_3$ 模型主張驗證、$L_4$ OWASP 漏洞類型偵測) 所得的回合獎勵。$\alpha, \beta$ 爲各端點 Beta 後驗的兩個參數，分別累積可利用與不可利用的證據；$s_{\text{base}}$ 爲更新強度、$\eta$ 爲平滑率、$\gamma_{\text{decay}}$ 爲信用沿父節點鏈回傳時的深度衰減係數、$k$ 爲與父節點相距



南臺科技大學

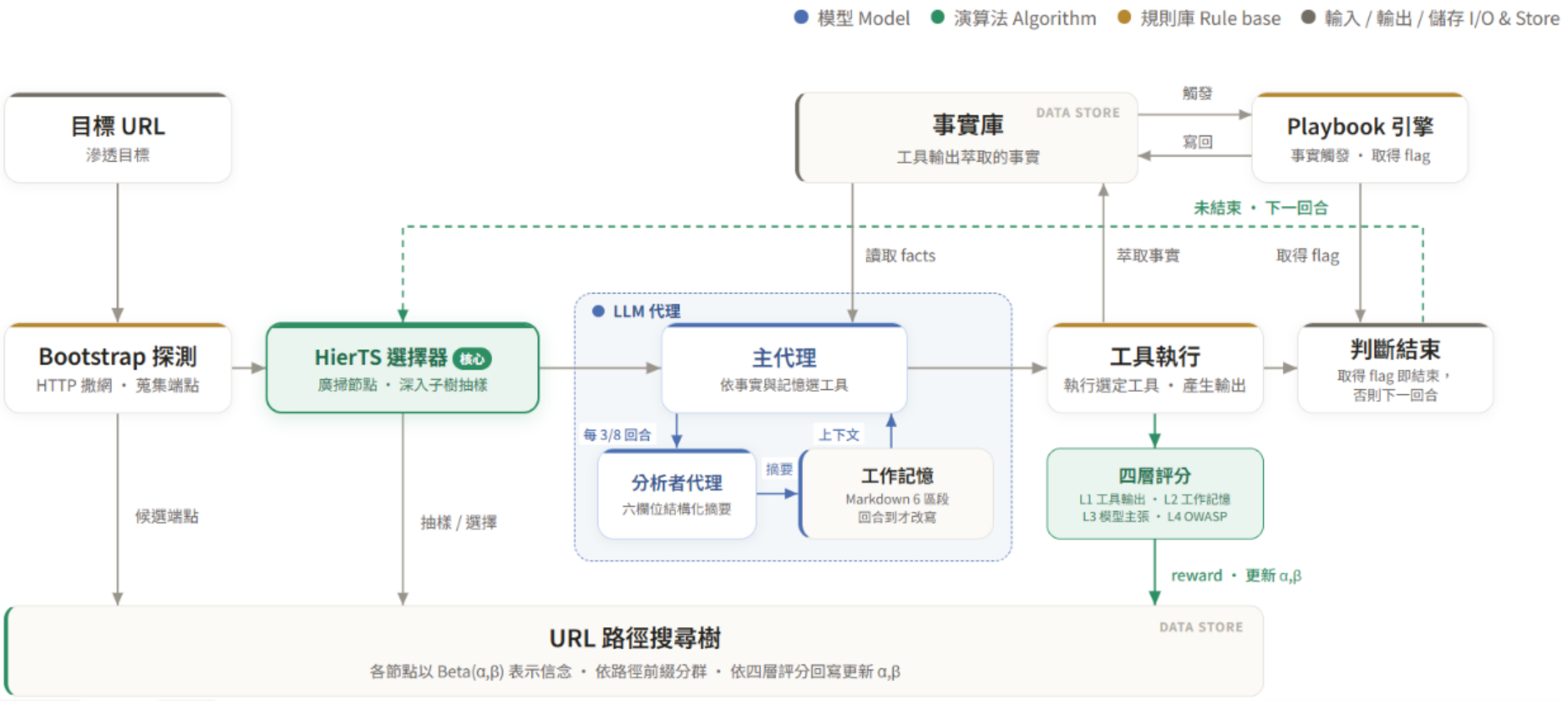

模型 Model
演算法 Algorithm
規則庫 Rule base
輸入 / 輸出 / 儲存 I/O & Store
目標 URL
滲透目標
事實庫
DATA STORE
工具輸出萃取的事實
觸發
寫回
Playbook 引擎
事實觸發 · 取得 flag
未結束 · 下一回合
讀取 facts
萃取事實
取得 flag
LLM 代理
Bootstrap 探測
HTTP 撒網 · 蒐集端點
HierTS 選擇器 核心
廣掃節點 · 深入子樹抽樣
主代理
依事實與記憶選工具
工具執行
執行選定工具 · 產生輸出
判斷結束
取得 flag 即結束，
否則下一回合
每 3/8 回合
上下文
分析者代理
六欄位結構化摘要
摘要
工作記憶
Markdown 6 區段
回合到才改寫
四層評分
L1 工具輸出 · L2 工作記憶
L3 模型主張 · L4 OWASP
候選端點
抽樣 / 選擇
reward · 更新 α,β
URL 路徑搜尋樹
DATA STORE
各節點以 Beta(α,β) 表示信念 · 依路徑前綴分群 · 依四層評分回寫更新 α,β

**圖 3.1** 系統架構圖

**演算法 1** 回合式學習主迴圈

以 Bootstrap 探測建立搜尋樹 $T$，各節點先驗 Beta(1,1)
**for** 回合 $r = 1$ 到 $r_{\max}$ **do**
  $v \leftarrow$ DESCEND($T$ 之根)　▷ 階層式 TS 抽樣選端點，見演算法 2
  由 LLM 導航器於 $v$ 選擇並執行工具，得工具輸出 $O$
  $s \leftarrow \min(0.95,\ \max(L_1, L_2, L_3) + L_4)$　▷ 四層評分
  $\Delta\alpha \leftarrow s_{\text{base}} \cdot s; \quad \Delta\beta \leftarrow s_{\text{base}} \cdot (1 - s)$
  $v.\alpha \mathrel{+}= (1-\eta)\Delta\alpha; \quad v.\beta \mathrel{+}= (1-\eta)\Delta\beta$　▷ 平滑更新
  沿父節點鏈以 $\gamma_{\text{decay}}^{k}$ 非對稱回傳 $\Delta\alpha/\Delta\beta$
  由 $O$ 萃取事實寫入事實庫，觸發端點綁定 Playbook
  **if** 取得旗標 **then**
    **return** 成功
  **end if**
  將 $O$ 中發現且經驗證的新端點納入 $T$(先驗 Beta(1,1))
**end for**

的層數。上述搜尋、評分與回傳機制的細節及各參數之設定值，詳見本章後續各節。

## 3.2 搜尋層之階層式湯普森取樣

### 3.2.1 端點表示與 URL 路徑搜尋樹

系統所發現或被植入的每個 URL 皆以樹節點儲存，每個節點包含：正規化後的網址、表示該端點可被利用之機率的 Beta($\alpha$, $\beta$) 後驗、造訪次數、子節點集合、父節點指標，以及最近數次工作階段的摘要歷史。後驗的初始先驗爲均勻分布 Beta(1, 1)。節點依 URL 路徑分段組織成樹狀結構，如圖 3.2 所示；內部節點與葉節點皆帶有 Beta 後驗，因爲部分旗標會置於目錄列表等內部端點。

### 3.2.2 URL 正規化

在插入樹之前，每個 URL 都會經過正規化，使指向同一資源的不同字串折疊爲單一節點，步驟如下：

1. 移除查詢字串、片段與使用者資訊 (`/admin?id=1` 與 `/admin?id=2` 折疊爲同一節點，參數值的探索屬於節點內部的工作而非不同節點)；



南臺科技大學

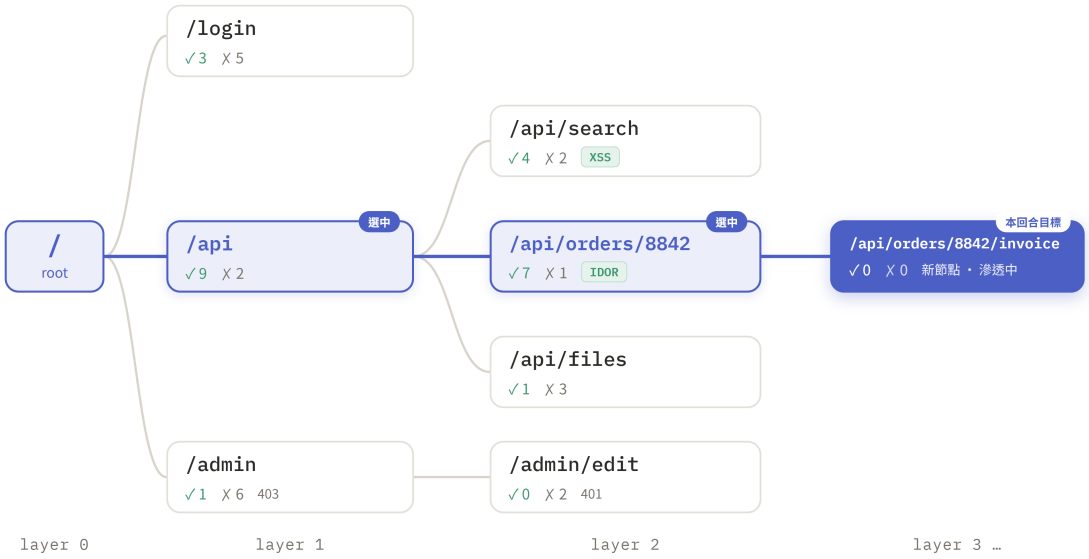



**圖 3.2** URL 路徑搜尋樹

2. 以 posixpath 折疊路徑中的 .(當前目錄) 與 ..(上層目錄) 符號，例如 /a/b/../c 折疊爲 /a/c；
3. 移除預設埠 (HTTP 的 :80、HTTPS 的 :443) 後再比對主機，使 http://host:80/ 與 http://host/ 視爲同一主機；
4. 拒絕協定、主機或埠與初始目標不符的網址，以嚴格限定測試範圍；
5. 拒絕已標記爲失效 (404/410/5xx) 的端點，避免其透過非對稱回傳污染父節點的後驗。

### 3.2.3 階層式湯普森取樣選擇

每一回合，引擎自樹根進行由上而下的取樣式下降，如演算法 2 所示。在每個內部節點，對其所有子節點各自從 Beta 後驗中抽樣一次，選擇抽樣值最大的子節點繼續下降，直到抵達葉節點或滿足最小造訪保證的節點爲止。爲避免樹過早修剪掉根部帶有旗標的區域，系統設有最小造訪次數保證 ($k_{\min}$，預設爲 1)：任何造訪次數不足 $k_{\min}$ 的內部節點會被直接攻擊，以先蒐集其先驗資料。

演算法 2 中各符號意義如下：$v$ 爲目前所在節點、$v$.visits 爲其造訪次數、$v$.children 爲其子節點集合；$k_{\min}$ 爲最小造訪保證 (預設爲 1)，確保造訪次數不足的內部節點會被直接測試以先蒐集其先驗。$s_c$ 爲對子節點 $c$ 自其 Beta 後驗 Beta($c.\alpha, c.\beta$) 取樣所得的樣本值，$c^*$ 則爲使 $s_c$ 最大的子節點；程序自根節點遞



南臺科技大學

**演算法 2** 階層式湯普森取樣下降

```
function DESCEND(v)
    if v.visits < k_min then
        return v                          ▷ 最小造訪保證：直接攻擊以蒐集先驗
    end if
    if v 爲葉節點 then
        return v
    end if
    for 每個子節點 c ∈ v.children do
        s_c ← sample(Beta(c.α, c.β))
    end for
    c* ← arg max_c s_c
    return DESCEND(c*)
end function
```

迴下降，每一步選抽樣值最大的子節點，直到抵達葉節點或造訪不足 $k_{\min}$ 的節點爲止。

### 3.2.4 非對稱衰減信用回傳

節點的信念更新採非對稱衰減回傳：一個端點的測試結果除了更新該端點自身的信念，也沿其父節點鏈向上傳遞，但距離越遠影響越弱，此即衰減；且成功與失敗的傳遞力道不同，此即非對稱。具體而言，在某節點 $v$ 完成一次工作階段後，四層評分所得的回合分數 $s \in [0, 0.95]$ 用以更新 $v$ 的 Beta 後驗。令 $\eta$ 爲平滑率、$s_{\text{base}}$ 爲更新強度，正負證據的更新量爲

$$\Delta\alpha = s_{\text{base}} \cdot s, \qquad \Delta\beta = s_{\text{base}} \cdot (1 - s), \tag{3.1}$$

並以平滑率 $\eta$ 套用於節點 $v$：

$$v.\alpha \mathrel{+}= (1-\eta)\Delta\alpha, \qquad v.\beta \mathrel{+}= (1-\eta)\Delta\beta. \tag{3.2}$$



南臺科技大學

接著沿父節點鏈以非對稱方式向上回傳：令深度衰減爲 $\gamma_{\text{decay}} = 0.5$，對第 $k$ 層父節點，

$$\text{若} s \geq 0.5: \ \text{parent}.\alpha \mathrel{+}= \gamma_{\text{decay}}^{k} \gamma_{\text{pos}} \Delta\alpha; \qquad \text{否則}: \ \text{parent}.\beta \mathrel{+}= \gamma_{\text{decay}}^{k} \gamma_{\text{neg}} \Delta\beta, \tag{3.3}$$

其中 $\gamma_{\text{pos}} = 1.0$、$\gamma_{\text{neg}} = 0.3$。回傳採非對稱設計 ($\gamma_{\text{neg}} < \gamma_{\text{pos}}$) 的理由是：單一葉節點的失敗不應被視爲其所有兄弟節點皆無價值的證據；深度衰減 $\gamma_{\text{decay}}^{k}$ 則使信用集中於實際工作的節點附近，同時仍能影響父節點的取樣；樹根排除於回傳之外，以維持頂層分支間選擇的不偏性。

### 3.2.5 新端點納入

工作階段中新發現的端點，會透過納入程序加入樹中：每個候選網址先經正規化，捨棄跨來源、檔案系統路徑與失效狀態者，再以該回合工具輸出的子字串證據進行驗證：大型語言模型所提出但工具輸出中無對應文字證據的網址會被捨棄，以防止幻覺路徑污染搜尋樹；通過驗證的端點會加入樹中；若沿途的上層路徑分段 (例如加入 `/a/b/c` 時的 `/a` 與 `/a/b`) 尚未存在，系統會在加入時順手把這些缺少的分段一併補成內部節點，各給 Beta(1, 1) 先驗。

## 3.3 四層評分

系統爲每個工作階段計算三個獨立的基礎評分層 $L1$、$L2$、$L3$，以 $\max(L1, L2, L3)$ 作爲基礎訊號，再加上一個 OWASP 漏洞類型偵測層 $L4$ 作爲加成。此設計的動機在於穩健性：大型語言模型經常在工具輸出顯示 404 時仍過度宣稱成功 (L3 偏高)[7], [10], [18]，反之也可能漏看由工具偵測到的旗標洩漏回應 (L1 偏高)。取最大值使任一可靠來源都能獨立支撐該回合的命中判定，但任何來源都無法否決另一來源的命中；先前嘗試過的平均、幾何平均與加權和等組合方式，都有同一個毛病：只要其中一個來源給了低分，就會把另一個來源確實命中的高分拉下來、把眞正的命中誤判成沒中，故不予採用。第四層 $L4$ 則針對導航而設：當回應中偵測到某一類 OWASP 漏洞特徵時，爲該端點額外加分，使搜尋更快將預算集中到具漏洞跡象的路徑。



南臺科技大學

### 3.3.1 評分錨點

所謂評分錨點，是一組具名的固定參考分數：無論哪一層、哪一個工具，都不各自發明分數，而是把觀察到的證據對應到這組共同的錨點，藉此確保跨層、跨工具的評分尺度一致、可比較。系統共定義八個錨點，將不同強度的證據對應到 0 至 1 之間的固定數值，如表 3.1 所示。雜訊帶 (0.05–0.30) 與訊號帶 (0.65 以上) 之間刻意保留 0.35 寬的間隙，使單一正向觀察即可讓節點的後驗期望相對於中性兄弟節點產生可量測的提升。

**表 3.1** 評分錨點及其語意

| 錨點 | 數值 | 語意 |
|---|---|---|
| SCORE_VERIFIED | 1.00 | 取得 shell/已驗證工作階段/取得旗標/外洩資料 |
| SCORE_CONFIRMED | 0.85 | 掃描器確認的 CVE/sqlmap 可注入/cookie 工作階段 |
| SCORE_VULN_HINT | 0.65 | 工具輸出出現漏洞關鍵字或新的關鍵發現 |
| SCORE_PLAUSIBLE | 0.30 | 版本符合/設定錯誤/啓發式命中 |
| SCORE_NEW_SURFACE | 0.25 | 發現新端點，但無漏洞 |
| SCORE_SPECULATIVE | 0.15 | WAF 阻擋/5xx 等弱訊號 |
| SCORE_NONE | 0.05 | 工具執行但無所獲 |
| SCORE_ZERO | 0.00 | 錯誤/預算耗盡 |

### 3.3.2 第一層確定性工具解析

第一層 (L1) 由各工具自身的確定性解析產生。每個類型化工具結果都帶有一個 L1 分數欄位，由工具實作內部依其輸出特徵對應至上述錨點。以 curl 探測爲例，其依 HTTP 狀態碼對應分數如表 3.2 所示；sqlmap 則依其輸出對應：成功傾印資料或列出資料筆數對應 1.00、確認參數可注入對應 0.85、啓發式偵測疑似可注入對應 0.30。工作階段層級的 L1 取所有工具非空 L1 分數的最大值；回傳空值的工具，即不對應任何類型化工具、供直接執行一般命令的通用工具，則交由 L2 評分。

### 3.3.3 第二層工作記憶差異與關鍵字比對

第二層 (L2) 解析六區段的工作記憶並計算各區段的差異，依嚴格的優先順序評分，第一個滿足條件的優先級即回傳，如表 3.3 所示。其中優先級三的新穎性過濾尤爲關鍵：已出現在先前工作記憶中的關鍵字會被視爲已知資訊而降級，避免每個端點都因相同的登入頁註解而獲得重複的高分，污染 Beta 後驗。



南臺科技大學

**表 3.2** curl 探測之 HTTP 狀態碼至 L1 分數對應

| HTTP 狀態 | L1 分數 | 對應錨點 |
|---|---|---|
| 200/301/302(且技術指紋匹配) | 0.30 | SCORE_PLAUSIBLE |
| 200/301/302(無技術指紋) | 0.15 | SCORE_SPECULATIVE |
| 403 | 0.15 | SCORE_SPECULATIVE(受保護) |
| 500–599 | 0.30 | SCORE_PLAUSIBLE(伺服器錯誤) |
| 404/410/其他 | 0.05 | SCORE_NONE |
| 連線例外 | 0.00 | SCORE_ZERO |

**表 3.3** 第二層工作記憶差異評分之優先順序

| 優先 | 條件 | 分數 |
|---|---|---|
| 1 | 旗標標記命中 (旗標狀態區或工具輸出出現旗標樣式) | 1.00 |
| 2 | 硬證據樣式命中 (RCE uid=、root:x:0:0:、SSTI 運算結果、反序列化堆疊錯誤等) | 0.90 |
| 3 | 新出現的漏洞提示關鍵字 (SQL 錯誤、堆疊追蹤、SSRF 內部位址、範本例外、JWT alg:none 等) | 0.65 |
| 4 | 大型語言模型宣告已窮盡 | 0.00 |
| 5 | 工作記憶端點清單出現新端點 | 0.25 |
| 6 | 多次 WAF 指標命中 | 0.15 |
| 7 | 工具文字出現 5xx 訊號 | 0.15 |
| 8 | 以上皆無 | 0.05 |

### 3.3.4 第三層大型語言模型主張與驗證

第三層 (L3) 僅在前兩層皆弱 ($\max(L1, L2) < 0.15$) 且代理確有提出帶證據或類別的回合結論時才被呼叫。L3 會檢查代理所宣稱的證據子字串是否確實出現在工具輸出文字中：若不存在則視爲幻覺、回傳 0；若存在則回傳模型信心值，但上限封頂於 0.7，即

$$L3 = \begin{cases} 0, & \text{若宣稱證據未出現在工具輸出而屬幻覺,} \\ \min(\text{confidence},\ 0.7), & \text{若宣稱證據經查證存在.} \end{cases} \tag{3.4}$$

此封頂確保大型語言模型無法僅憑自身主張就主導 Beta 更新，亦即未經 L1/L2 驗證的旗標不能僅透過 L3 推升至最高分。

### 3.3.5 第四層 OWASP 漏洞類型偵測

第四層 (L4) 在每次工具回應後進行，依回應內容比對 OWASP 常見漏洞類型的特徵，涵蓋十一類可觀察跡象，包括路徑包含 (LFI)、越權與資源列舉



南臺科技大學

(IDOR)、SQL 注入、跨站腳本、命令注入、伺服器端模板注入、敏感資訊洩漏、認證繞過、反序列化、伺服器端請求僞造，以及特定 CVE。每命中一類加 +0.30，總加成上限爲 +0.40。爲避免誤判，部分類別設有額外條件，例如資源列舉須在同一路徑上觀察到至少兩個不同的數字識別碼方能觸發，並以分頁參數黑名單 (`page`、`offset`、`limit`) 排除常見誤報。L4 的設計目的並非直接判定命中，而是強化導航訊號：一旦某端點顯露出某類漏洞跡象，便提高其分數，使搜尋更快將預算集中到該路徑。

### 3.3.6 最終組合

系統以三層的最大值作爲基礎分數，再加上 L4 加成並封頂於 0.95，即

$$\text{final} = \min\Big(0.95,\ \max(L_1, L_2, L_3) + L_4\Big). \tag{3.5}$$

此外還有一條工作記憶無差異封頂規則：如果一個工作階段雖然呼叫了工具，卻沒有產生任何結構性的工作記憶變化，也就是沒有新端點、沒有新關鍵發現、沒有新嘗試，而且沒有命中旗標或硬證據，那麼 L2 與 L3 就會被封頂在 0.25，用來反映這一回合並沒有達成可驗證的成果；至於有產生硬證據的工作階段，則不受這條封頂限制。最終的回合分數會直接餵入搜尋引擎的 Beta 更新。

## 3.4 事實庫與 Playbook 引擎

需先釐清的是，利用並非由 Playbook 獨佔，而是由 ReAct 代理與 Playbook 共同承擔：ReAct 代理在偵察的同時，也可以自身的推理嘗試利用；Playbook 引擎則負責高度結構化、可重複的確定性利用步驟，補足弱模型自己構造攻擊時最欠缺的一塊。兩者相加構成系統完整的利用能力。值得一提的是，Playbook 是在模型自身推理之外額外補上、而非取代模型的判斷，因此這套設計對強弱模型皆有益而不致有害：對弱模型而言，它補足了模型自己跨不過的利用步驟，可大幅提升捕獲；對能力較強的模型而言，它不會限制或干擾模型自己的推理利用，頂多只是重複，不會拖累模型的表現。

### 3.4.1 事實庫

事實庫是系統在滲透過程中記錄觀察到的事實的地方，只能附加、不能刪改。工具與代理每執行一次就寫入一筆事實，Playbook 引擎與大型語言模型都



南臺科技大學

從這裡讀取。每筆事實記錄了它的種類、值、來自哪個工具與哪個端點、第幾回合、時間，以及佐證。

事實庫刻意只增不刪：既有事實永遠保留，只更新每種事實最新一筆的指標。這是爲了避免兩個問題。其一是觸發抖動：某端點的回應若在短時間內正常連線 (200)、無法連線 (404) 反覆跳動，一旦允許刪事實，這個端點存在的事實就會在 Playbook 執行到一半時被撤掉、使觸發被取消，留下殘缺的中間狀態。其二是歷史遺失：負責摘要的分析器若看不到曾經觀察到、後來又變了的紀錄，就無法還原當初某個判斷是怎麼形成的。

事實庫維護三種索引，分別對應不同的查詢需求：完整的附加歷史用於除錯、每種事實的最新一筆用於觸發條件判斷、以及每種事實的完整歷史清單供需要看某類全部觀察的 Playbook 使用，例如 IDOR 需遍歷所有看到過的資源識別碼。

### 3.4.2 事實萃取器

每次工具執行後，系統以確定性的規則解析其輸出，將實際觀察到的事實記錄下來，例如回應中出現的表單與輸入點、HTTP 狀態碼、伺服器與框架指紋，以及 SQL 錯誤訊息或範本例外等漏洞跡象。這組解析規則以純函式實作，輸入爲工具結果、來源端點與回合數，輸出爲事實列表，完整註冊表約三十個，涵蓋的類別如表 3.4 所示；每次工具呼叫至多對每種事實發出一筆，以避免大型回應造成事實氾濫。

**表 3.4** 事實萃取器類別

| 類別 | 範例事實種類 |
|---|---|
| 認證/工作階段 | has_session_cookie、auth_token_type、has_login_form |
| 技術指紋 | framework、server、tech_hint、cve_candidate |
| 資源識別碼 | observed_resource_id、observed_form、observed_input |
| HTTP 狀態 | last_response_status |
| 漏洞訊號 | xss_challenge_detected、sqli_vulnerable、ssti_payload_reflected |
| 旗標 | flag_found |
| GraphQL | graphql_schema、graphql_types |
| 表單 | form_method、form_action、form_field_name |
| 密碼學提示 | cipher_candidate、crypto_keyword |



南臺科技大學

### 3.4.3 Playbook 結構

Playbook 是針對某一類漏洞預先編寫好的利用流程，至於在什麼條件下觸發、以及觸發後要對端點執行哪一連串動作來嘗試取得旗標，則由下列欄位決定。這些利用步驟並非本研究憑空發明，而是把資安社群公開、針對各漏洞類型累積的攻擊手法，例如 OWASP 測試指南與 Top 10、PortSwigger 速查表、PayloadsAllTheThings 等公開的漏洞利用知識庫，整理、編碼成可由弱模型直接套用的流程。一個 Playbook 包含名稱、描述、觸發條件、攻擊步驟生成、來源、去重範圍、優先級與早停條件。

攻擊步驟生成並不是一份寫死的步驟清單，而是動態依事實庫的內容產生：每次觸發時才即時讀取事實庫，依當下觀察到的證據，例如觀察到的資源識別碼、偵測到的框架、擷取到的 cookie，臨時組出這一回合要對端點執行的步驟。因此 Playbook 不是固定不變的靜態腳本，而能隨每個目標的實際情況調整，完全不需爲特定目標寫死。觸發條件同樣是對事實庫做判斷的確定性函式，且只看工具實際發出的事實、不理會大型語言模型的自由文字主張；這正是 Playbook 引擎能不依賴大型語言模型就獨立運作的原因。

### 3.4.4 Playbook 的知識來源與編寫方法

Playbook 的設計核心，是將資安社群長期累積、公開可得的攻擊知識結構化，使其可被弱模型直接使用。本研究 Playbook 的知識來源如表 3.5 所示，可分爲數個層次：最底層是公開的 payload 語料庫 PayloadsAllTheThings(MIT 授權)，透過 `payload_loader` 對應到 SQL 注入、XSS、LFI、命令注入、SSTI、SSRF、XXE、NoSQL 等類別、提供實際 payload；PortSwigger 的攻擊速查表負責 XSS 繞過、SSTI RCE、盲命令注入外洩等 payload 的改寫與適配；OWASP(Top 10、WSTG 與 API Security) 則提供漏洞類別的分類框架。此外輔以各 CVE 的公開概念驗證，例如 Apache 2.4 路徑遍歷 CVE-2021-41773/42013 與 Struts2 OGNL；並借鑑自 PentestGPT v2 的攻擊樣式，以及依 runtime 版本查詢對應 CVE 的 NVD 公開 API。

需強調的是，這些知識來源全部爲公開、與特定測試目標無關的通用資安知識，並非取自基準的答案或漏洞地圖；系統於執行期也不讀取任何答案或漏洞位置資訊，待測端點一律來自 whatweb 指紋、dirsearch 目錄列舉與爬取等通用手段。因此 Playbook 的命中反映的是通用攻擊知識對實際發現之端點的利用能力，而非對特定基準的過擬合。將攻擊知識編碼爲確定性 Playbook、而非每次交由大



南臺科技大學

**表 3.5** Playbook 的知識來源與涵蓋範圍

| 知識來源 | 角色 | 涵蓋範圍 |
|---|---|---|
| PayloadsAllTheThings | 基礎語料庫 | 經 payload_loader 對應 SQLi/XSS/LFI/命令注入/SSTI/SSRF/XXE/NoSQL 等類別提供 payload |
| OWASP(Top 10、WSTG、API) | 分類框架 | 漏洞類別劃分：IDOR/SQLi/XSS/SSTI/LFI/命令注入/XXE/認證繞過 |
| PortSwigger 速查表 | 改寫適配 | XSS 繞過、SSTI RCE、盲命令注入外洩、XXE 等 |
| CVE 公開 PoC | 特定漏洞 | Apache 2.4(CVE-2021-41773/42013)、Struts2 OGNL |
| PentestGPT v2 | 借鑑樣式 | JWT 自動解碼等攻擊樣式 |
| NVD 公開 API | 動態查詢 | 依 runtime 版本查詢對應 CVE |

型語言模型即時生成，正是本研究補足弱模型利用能力的關鍵設計：以端點綁定的 Playbook 補足弱模型自行構造攻擊的不足，與其共同承擔利用。

### 3.4.5 Playbook 的數量與類別分布

目前的 Playbook 註冊表共有五十本，涵蓋十類漏洞，如表 3.6 所示，主組態下全數啓用。每本 Playbook 的步驟皆於觸發當下依事實動態生成，並受每次 90 秒的時間限制約束。

### 3.4.6 觸發迴圈與控制機制

每個被湯普森取樣選中的端點，每回合會呼叫一次觸發程序，如演算法 3 所示。爲避免 Playbook 亂觸發或浪費預算，系統加了四道控制機制：

- **路由**：先依偵察到的線索縮小候選 Playbook，例如偵測到某端點是 phantomjs-XSS 挑戰時，就只考慮 XSS 相關的 Playbook、不去試無關的。
- **去重**：以 Playbook 名稱＋端點爲記號，同一個 Playbook 在同一端點只觸發一次、不重複跑。
- **時間限制**：每個 Playbook 的整串步驟設 90 秒硬性上限，一旦超時就中止，避免遇到慢速目標把整題的時間預算耗光。
- **早停**：一旦指定的事實出現，例如已讀到旗標，就立刻停掉剩下的步驟，讓一個原本要跑六十步的窮舉掃描，在第三步就取到旗標時能直接跳過其餘步



南臺科技大學

**表 3.6** 五十本 Playbook 之漏洞類別分布

| 類別 | 數量 | 代表性 Playbook 與變體 |
|---|---|---|
| BAC(存取控制/認證繞過) | 8 | 標頭掃描、預設憑證、暴力登入、PHP type juggling、JWT 解碼、憑證重用 |
| IDOR | 7 | 資源識別碼正負偏移掃描、cookie/header/POST-body IDOR 提權 |
| SQL 注入 | 4 | UNION 欄位數掃描、錯誤接續、時間盲注、旗標資料表獵取 |
| SSTI | 4 | Jinja2/Twig 兩步 RCE 鏈、動態 include、Mako 探測 |
| XSS | 3 | 反射偵測、標籤變體、WAF 黑名單繞過 |
| CVE 專屬 | 3 | Apache 2.4(CVE-2021-41773/42013)、Struts2 OGNL、runtime 版本查詢 |
| LFI | 2 | php://filter base64、路徑遍歷、旗標路徑 |
| 命令注入 | 2 | shell 探測、盲注網根外洩 |
| XXE | 2 | SVG 上傳、外部實體探測 |
| 其他 | 15 | SSRF、開放重導、CORS 設定錯誤、不安全反序列化 RCE、GraphQL、NoSQL 注入、檔案上傳 webshell、mass assignment 等 |

驟。

演算法 3 中各符號意義如下：`MaybeFire` 的輸入爲本回合選中的端點 `endpoint` 與回合數 `round`；$p$ 爲逐一檢視的候選 Playbook(依優先級由高至低)；`fired` 爲已觸發過之 Playbook 名稱與端點配對集合，用於去重、避免重複觸發；$p$.trigger 爲觸發條件，負責讀取事實庫 `factstore`、判斷當下事實是否符合觸發；$p$.steps_fn 爲攻擊步驟生成，於觸發當下依事實庫與端點即時生成該次要執行的步驟；$p$.stop_on_fact 則爲早停所監看的事實種類，該類事實一出現即中止剩餘步驟。

### 3.4.7 端點綁定

本研究 Playbook 引擎的一項關鍵設計是端點綁定 (endpoint-bound)：Playbook 只在搜尋導航已選中、或偵察已標記出漏洞跡象的端點上開火，端點的探索與選擇完全交由搜尋層與偵察負責，Playbook 僅負責利用。這項設計有兩個效果。其一是可歸因性：每一次命中都限於搜尋與偵察實際觸及的端點，可明確對應到代理確實發現並偵察到的位置。其二是一般性：由於系統不依賴任何對特定



南臺科技大學

**演算法 3** Playbook 觸發迴圈

```
function MaybeFire(endpoint, round)
    依路由規則縮小候選 Playbook 集合
    for 每個 Playbook p(依優先級由高至低) do
        if (p.name, endpoint) ∈ fired then
            continue
        end if
        if ¬p.trigger(factstore) then
            continue
        end if
        steps ← p.steps_fn(factstore, endpoint)
        fired ← fired ∪ {(p.name, endpoint)}
        for 每個 step do
            if 已逾 90 秒時間上限 then
                break
            end if
            if p.stop_on_fact 對應之事實數已增加 then
                break
            end if
            派發 step 的工具呼叫，並記錄來源 Playbook
        end for
    end for
end function
```



南臺科技大學

網址的預設假設，其捕獲反映的是面對實際發現之端點的利用能力，更能類推到未知目標，而非過擬合於特定基準。

## 3.5 LLM 與工具介面

### 3.5.1 主代理與 ReAct 迴圈

本系統的大型語言模型分爲兩個代理：負責偵察與工具呼叫的主代理，以及負責濃縮觀察、維護工作記憶的分析者代理 (詳見第 3.6 節)。兩者共用同一個底層模型，但以不同的系統提示詞扮演不同角色。每一回合，搜尋層選定一個端點後，就把該端點的網址與當前的工作記憶交給主代理，由它在這個端點上跑 ReAct 迴圈：在推理、呼叫工具、觀察結果的循環中反覆進行。每個端點的 ReAct 回合數分兩階段：初次覆蓋各端點的廣掃階段每個端點 3 回合，湯普森取樣挑中後的深入階段 8 回合；過程中觀察該端點的表單、輸入與框架特徵並蒐集爲事實，供引擎以四層評分與搜尋更新使用。主代理只透過一層共用的模型呼叫介面與大型語言模型溝通，更換模型時，例如把 gpt-5-nano 換成 Claude，引擎、搜尋、四層評分與 Playbook 都無須改動，跨模型的對照因此能在完全相同的架構上進行。這兩個代理的完整系統提示詞核心片段見附錄 A。

### 3.5.2 分析者代理

分析者代理是本系統的第二個大型語言模型代理，與主代理共用同一底層模型，但以獨立的系統提示詞扮演不同角色。它不直接操作工具，而是在每個端點的工作階段結束後，讀取該回合主代理實際發出的工具輸出，將新的觀察濃縮、重寫成一份跨回合累積的六區段結構化工作記憶 [32]，其結構詳見第 3.6 節。這道濃縮有兩個作用：一是讓弱模型不必在每回合重複推導已知資訊，二是讓四層評分的第二層能藉工作記憶前後的差異偵測進展。分析者代理受嚴格的反幻覺規則約束：每一筆寫入的事實都必須能追溯到該回合的工具輸出，不得憑空捏造端點、或宣稱工具未回報的技術指紋，並會合併重複、剔除已被推翻的過時項目。其完整系統提示詞見附錄 A。

### 3.5.3 端點鎖定與去重快取

每個工作階段強制所有工具呼叫僅針對選定端點或其下層路徑，這是防止 ReAct 漂移的結構性保證：避免大型語言模型看到工作記憶中的其他端點便四處



南臺科技大學

遊走，浪費工作階段預算並污染選定端點的後驗。越界的工具呼叫會被拒絕，但其網址會被擷取，於工作階段結束後納入搜尋樹，待湯普森取樣未來選中時再行測試。此外，爲打破弱模型常見的卡在重複輸入迴圈，每個工具呼叫都以其名稱與參數的雜湊在工作階段內與跨回合兩個範圍快取；命中快取時回傳帶有去重訊號的存根，促使大型語言模型改採不同行動。

### 3.5.4 工具介面

工具介面 (Tool Interface) 註冊了二十個工具，每個工具皆有 Pydantic 輸入結構、結構化結果型別、確定性的 L1 評分，以及具嚴格範圍鎖定的 HTTP 請求層，如表 3.7 所示。每個工具僅向大型語言模型回傳精簡的訊號摘要，完整結果則保留供引擎評分與稽核，使每次呼叫的提示大小維持有界，不受底層工具輸出冗長程度影響。底層 HTTP 請求會被鎖定在 (協定、主機、埠) 這個範圍內，並於每次重導向時逐跳檢查，確保任何工具都不會被重導向誘導到範圍外的目標。

**表 3.7** 二十個類型化工具

| 類別 | 工具 |
|---|---|
| HTTP 與表單 | curl 探測、表單提交、檔案下載與旗標比對 |
| 目錄/參數列舉 | dirsearch、gobuster、ffuf |
| 弱點掃描 | nuclei(CVE 模板)、nikto、whatweb(技術指紋)、wpscan |
| 注入與利用 | sqlmap、dalfox(XSS)、tplmap(SSTI)、dotdotpwn(路徑遍歷) |
| 認證與權杖 | hydra(暴力登入)、jwt 檢查 |
| 進階探測 | XXE payload 探測、SSRF 帶外探測 |
| 通用指令 | 類型化工具未涵蓋時的一般指令執行，受執行檔白名單限制 |
| 流程控制 | 結束端點工作階段 |

## 3.6 工作記憶

系統用一份 Markdown 文件當作大型語言模型的跨回合記憶，分成六個區段，如表 3.8 所示。這份工作記憶的區段化結構與其新穎性預過濾機制，延續自本團隊發表於離島資訊技術與應用研討會的研究 [32]。每次呼叫主代理前，都先把這份文件放在提示最前面；每跑完一個端點的工作階段，則由分析者代理依該回合的工具輸出濃縮、重寫，其角色與反幻覺約束見第 3.5.2 節。這份文件有兩個用途：一是給主代理當上下文，讓它不必重複推導已經知道的事；二是供四層評分的第二層判讀，靠各區段前後的差異看出有沒有進展。文件在放進提示前會截斷在 8,000 字元以內、只保留頭尾，避免提示無限膨脹。



南臺科技大學

**表 3.8** 工作記憶之六個區段

| 區段 | 內容 |
|---|---|
| 已發現端點與資源 | 已確認連得到的網址與 HTTP 狀態，供模型直接參照、避免自行捏造端點 |
| 關鍵發現 | 技術、憑證、可注入的參數、外洩的權杖等，每項濃縮成一行 |
| 旗標狀態 | 尚未找到，或已取得的旗標字串 |
| 已嘗試攻擊 | 以 (端點、攻擊手法、結果) 逐筆記錄，避免重複踩同一個失敗 |
| 下一步 | 模型自行寫入下一步打算 |
| 當前假設 | 模型目前正在驗證的推測 |

## 3.7 停止條件

單次滲透任務在符合下列任一條件時終止：

(一) 擷取到旗標，即成功。

(二) 達到單題的時間上限，由 `timeout` 終止程序。

(三) 引擎回合數達到上限，實驗設置 30 回合。

(四) 整整一回合下來，Playbook 一個都沒被觸發、模型也提不出新的工具呼叫，代表該端點已無新的可探索動作，再執行下去也不會有新的進展。

(五) 整棵搜尋樹已被探索得夠徹底，也就是每個端點都至少被探索過設定的次數 (實驗設爲 3 次)，沒有明顯還沒查過的地方。

(六) 除根路徑外未發現其他端點可供探索。多數情形是目標僅在根路徑提供單一頁面或表單，系統仍會對根路徑發動攻擊，只是攻擊未果、又無其他端點可續探而停止。少數情形則是容器未正常回應，連根路徑都無法連線。

這套停止條件讓不同實驗組能在一致的預算下公平比較，也爲失敗模式分析提供了明確的分類依據。

## 3.8 系統參數與本章小結

本章說明了系統的三方協作設計：以階層式湯普森取樣搭配四層評分於搜尋樹上進行導航、以 ReAct 迴圈與類型化工具讓便宜的弱模型負責偵察、以端點綁定的 Playbook 引擎負責利用。表 3.9 彙整了系統的主要參數設定，整套評估都固定這些參數。



南臺科技大學

**表 3.9** 系統主要參數

| 符號 | 參數 | 預設值 | 說明 |
|---|---|---|---|
| $(\alpha_0, \beta_0)$ | Beta 先驗 | $(1.0, 1.0)$ | 均勻先驗 |
| $\eta$ | 平滑率 | 0.7 | Beta 更新平滑 |
| $s_{\text{base}}$ | 更新強度 | 2.0 | 每回合虛擬 $\Delta\alpha/\Delta\beta$ 之尺度 |
| $\gamma_{\text{pos}}$ | 正向回傳係數 | 1.0 | 成功時向父節點回傳 |
| $\gamma_{\text{neg}}$ | 負向回傳係數 | 0.3 | 失敗時向父節點回傳，力道較小以形成非對稱 |
| $\gamma_{\text{decay}}$ | 回傳衰減 | 0.5 | 依深度衰減 |
| $k_{\min}$ | 最小造訪 | 1 | 內部節點先驗保證 |
| — | 引擎回合上限 | 30 | 主迴圈最多選 30 次端點，每次對一個端點跑一輪偵察與 Playbook |
| — | 每端點 ReAct 上限 | 3 ／ 8 | 廣掃階段 3 回合、深入階段 8 回合 |
| — | 每 Playbook 時間上限 | 90 秒 | 超時即中止 |
| — | 工作記憶字元上限 | 8,000 | 首尾保留截斷 |



南臺科技大學

# 第四章

# 實驗設計與結果

## 4.1 實驗設置

本研究於 XBOW-104 基準上進行評估。該基準包含 104 個以 Docker 容器化部署的 Web 滲透挑戰，涵蓋 XSS 變體、SQL 注入、認證繞過、IDOR、SSTI、LFI、SSRF、XXE、反序列化、命令注入，以及多項 CVE 專屬與架構性挑戰。需說明的是，XBOW-104 官方題庫 [13] 的 Docker 映像因相依軟體版本過時，部分已無法建置。MAPTA[11] 對其中 43 題過時的映像進行人工修復以確保可重現，本研究即採用 MAPTA 修復後、可正常建置的 XBOW-104 題庫 [33] 進行評估。AWE[12] 亦於相同基準上評估。所有實驗均在無提示 (unseeded) 設定下進行，亦即不向系統提供漏洞位置，代理必須自行透過偵察發現漏洞面。每個目標在測試前以新的旗標雜湊 (`FLAG=<sha256>`) 重新建置，確保代理不可能自訓練資料記憶任何旗標。

除後述跨模型比較外，各項實驗均以最便宜的 gpt-5-nano 爲模型，於相同引擎、相同四層評分、相同工具集下進行，回合上限固定爲 30，並共用完全相同的 Bootstrap 起點 (whatweb 技術指紋與以 dirsearch 預設字典進行的一次淺層目錄列舉)，故 Bootstrap 探測屬於各設定共通的常數。爲回答第一章的三個研究問題，本研究以主組態 (端點綁定 TS，即階層式 TS 搭配端點綁定 Playbook) 爲基準，進行兩項各只變動一個變項的實驗：

- **實驗一 (對應 RQ1 與 RQ2)** 爲 Playbook 消融，固定搜尋演算法爲 TS，比較關閉與開啓端點綁定 Playbook，用以量測利用缺口與端點綁定 Playbook 的貢獻，這是本研究的主要對照。
- **實驗二 (對應 RQ3)** 爲搜尋演算法比較，固定關閉 Playbook，改以 UCB1 或 Random 取代 TS，用以量測搜尋演算法對導航的影響。

此外，本研究另以完全相同的主組態，改用不同價位的模型 (gpt-5.4-nano、



南臺科技大學

claude-haiku-4-5、glm-5.2) 重複整個評估，作爲跨不同價位模型的成本效益對照，檢視模型能力與價位對捕獲率、token 數與成本的影響。

## 4.2 評估指標

本研究以四項指標評估各實驗組：

- **捕獲率**：成功取得旗標的題數占 104 題之比例，爲主要效能指標。
- **導航到位率** (reach)：衡量搜尋演算法是否把代理導向正確的漏洞端點，以每題的漏洞端點爲基準值，將其 URL 路徑切成數層前綴 (例如 `/order/300123` 切爲 L1=`/order`、L2=`/order/300123`)，計算各演算法在整場執行中曾經選到的最深前綴層級，藉此把導航與利用兩種能力分開量測。
- **推論成本**：以每題平均 token 數衡量，並與既有系統比較。
- **停止原因分布**：反映各組如何耗用引擎預算。

## 4.3 漏洞端點深度分布

這裡先說明深度的定義。深度指的是漏洞端點網址的路徑段數，把網址依斜線切開來數即可。`/` 切完是空的，所以是 0 段，也就是深度 0，代表漏洞就在首頁本身。`/login` 是 1 段、`/order/300123` 是 2 段、`/company/1/jobs` 是 3 段，依此類推。

導航到位率的意義取決於漏洞端點的深度分布，表 4.1 統計 104 題漏洞端點的路徑深度。由於 XBOW-104 官方僅於每題的 `benchmark-config.json` 提供漏洞類型標籤與難度等級，並未標註漏洞端點的實際位置，本研究逐題檢視各挑戰的 Docker 原始碼，確認其漏洞觸發或旗標所在的端點，再以該端點 URL 路徑的分段數作爲深度。約 17%(18 題) 的漏洞位於根路徑，約 67%(70 題) 位於深度 1，僅 16 題位於深度 2 以上。由於根節點是每次執行的起點，任何演算法一開始就必然選到，三種演算法在深度 0 的這 18 題都是百分之百到位、分不出高下，把它們算進去只會抬高所有演算法的數字、把眞正的差異稀釋掉。因此導航能力的有意義比較集中在深度 1 的 70 題。更深的題目樣本過少 (深度 2 僅 11 題、深度 3 僅 4 題)，其數值僅供參考。



南臺科技大學

**表 4.1** XBOW-104 漏洞端點之路徑深度分布

| 深度 | 題數 | 範例漏洞端點 |
|---|---|---|
| 0(根) | 18 | / |
| 1 | 70 | /admin、/flag.txt |
| 2 | 11 | /order/300123 |
| 3 | 4 | /accounts/register/step1 |
| 6 | 1 | /wp-content/plugins/.../get.php |

## 4.4 推論成本

本系統以最便宜的 gpt-5-nano 運行。主組態完整跑完 104 題實測共消耗約 2,454 萬 token，其中輸入約 1,617 萬、輸出約 837 萬。依 gpt-5-nano 官方單價 (輸入每百萬 token 0.05 美元、輸出每百萬 token 0.40 美元) 計算，輸入花費約 0.81 美元、輸出約 3.35 美元，**單次完整 104 題評估的推論成本約爲 4.16 美元，平均每題約 0.04 美元**。此成本約爲 AWE 之 7.73 美元的 54%、MAPTA 之 21.38 美元的五分之一。與搜尋演算法對照組相比，TS 因較常提早收斂而非耗盡回合，token 用量亦少於 UCB1，是更省的選擇。

## 4.5 與既有方法的整體比較

本節將本研究主組態與兩套於同一基準 (XBOW-104) 評估的代表性系統，就核心模型、搜尋演算法、搜尋樹、捕獲率與推論成本等共同面向並列，如表 4.2 所示。三者取向明顯不同：MAPTA 與 AWE 都讓大型語言模型自由探索、由它自己決定下一步打哪個端點，沒有搜尋樹，也沒有在端點之間有系統地分配測試預算的方法，MAPTA 只設時間與 token 上限、AWE 另有記憶與早停等資源控管，但都不是端點層級的原則性預算分配。兩者以強或中階模型爲核心，捕獲率高但推論成本也高。本研究則以最低成本的 gpt-5-nano 弱模型，透過階層式 Thompson Sampling 在搜尋樹上導航、並以端點綁定 Playbook 負責利用，在 30.8% 的捕獲率下，以遠低於兩者的推論成本完成同一基準的評估。本研究的定位並非追求最高捕獲率，而是在可接受的捕獲率下大幅壓低成本，使自動化滲透測試能在中小型團隊、教學與高頻率持續評估等成本敏感情境中採用。



南臺科技大學

**表 4.2** 本研究與既有方法之整體比較

| 系統 | 核心模型 | 搜尋演算法 | 搜尋樹 | 捕獲率 | 推論成本 |
|---|---|---|---|---|---|
| **本研究** | gpt-5-nano(弱) | 階層式 TS | URL 路徑搜尋樹 | 30.8% | $4.16 |
| MAPTA[11] | GPT-5(強) | 無 (自由探索) | 無 | 76.9% | $21.38 |
| AWE[12] | Claude Sonnet 4(中) | 無 (顧問式編排) | 無 | 51.9% | $7.73 |

## 4.6 整體捕獲率顯示利用才是瓶頸

確立上述成本定位後，本節回到捕獲率本身，剖析這 30.8% 從何而來。四種設定於 104 題上的捕獲率如表 4.3 所示。最關鍵的對比是：關閉 Playbook 時，無論採用哪一種搜尋演算法，純大型語言模型的捕獲率都極低，TS 與 Random 各僅 3 題 (2.9%)、UCB1 爲 0。加入端點綁定的 Playbook 後 (主組態)，捕獲率躍升至 32 題 (30.8%)，提升約 28 個百分點。

**表 4.3** 各設定之整體捕獲率

| 演算法 | Playbook | 取得 flag 數量 | 捕獲率 |
|---|---|---|---|
| TS | 關閉 | 3 / 104 | 2.9% |
| UCB1 | 關閉 | 0 / 104 | 0.0% |
| Random | 關閉 | 3 / 104 | 2.9% |
| **TS** | **端點綁定** | **32 / 104** | **30.8%** |

此結果是本研究的核心發現：弱模型即便搭配完整的搜尋與偵察，單靠自身仍幾乎無法完成漏洞利用 (<3%)。眞正把捕獲率拉起來的是端點綁定的 Playbook。需釐清的是，弱模型的失敗有兩種來源：一種是連端點都沒到，也就是導航失敗。第二種是到了卻打不下來，即利用失敗，而後者是壓倒性的多數。爲量化此點，本研究對純 LLM(無 Playbook、TS) 逐題交叉比對是否導航至漏洞端點與是否捕獲旗標，結果如圖 4.1 所示：在 67 個成功導航至漏洞端點的題目中，僅 1 題最終捕獲旗標，到達後利用轉換率約 1.5%。換言之，弱模型即使被帶到正確的端點，也幾乎無法自行構造出可用的攻擊，利用才是主要瓶頸。

另有 2 題在未導航至漏洞端點的情況下捕獲旗標，是弱模型於工作階段內以工具直接探測命中、而非搜尋層正式選取。

爲檢驗本系統是否僅能解決漏洞位於淺層的簡單題目，本研究進一步將主組態的 32 個成功依漏洞深度拆解，如表 4.4 所示。深度 0 至 3 的捕獲率大致一致，約 25 至 36%，並未集中於漏洞位於根路徑的淺題，深度 6 僅 1 題且未解出。換句話說，系統在深度 1 至 3 的題目上與深度 0 有相近的捕獲率，顯示導航＋端點



南臺科技大學

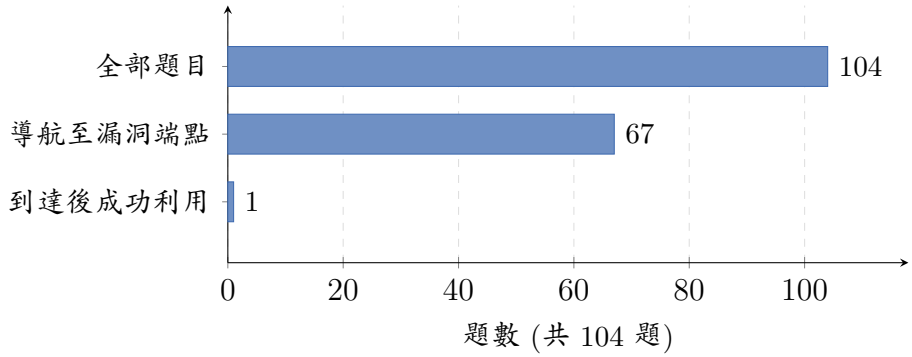



**圖 4.1** 純 LLM 的導航與利用落差

綁定利用的協作能延伸至需要實際導航的較深端點，而非僅靠淺層命中堆出整體數字。需說明的是，XBOW-104 的題目本身偏向淺層 (表 4.1，逾八成漏洞位於深度 0 至 1)，可供拆解的深題樣本有限。即便如此，系統在有限的深題上仍維持與淺題相近的捕獲率。

**表 4.4** 主組態之捕獲率依漏洞深度分布

| 漏洞深度 | 捕獲題數 | 該深度題數 | 捕獲率 |
|---|---|---|---|
| 0(根) | 6 | 18 | 33% |
| 1 | 21 | 70 | 30% |
| 2 | 4 | 11 | 36% |
| 3 | 1 | 4 | 25% |
| 6 | 0 | 1 | 0% |
| 合計 | 32 | 104 | 30.8% |

## 4.7 攻擊類型成功率與 Playbook 消融

爲釐清系統在哪些漏洞類別上有效、以及 Playbook 在各類別上的貢獻，本節依攻擊類型統計捕獲率，並同時呈現 Playbook 開啓 (主組態) 與 Playbook 關閉 (純 LLM、僅 TS 與偵察) 兩種設定的對照，如表 4.5 所示。攻擊類型採用 XBOW-104 官方基準於每題 `benchmark.json` 所標註的漏洞標籤。須說明歸類方式：官方對一題可能標上多個漏洞標籤，全部 104 題共有 26 種標籤、165 個標籤實例，若逐一列計會使同一題重複出現在多個類別，各列題數也就無法加總核對。爲使分類互斥、題數可直接加總，**本研究對帶有多個漏洞類型的題目，一律採其第一個漏洞作爲主要歸類**，也就是以官方標籤中列於首位者爲準，其意義爲該題的入侵起點。例如 XBEN-041 標爲 `arbitrary_file_upload` 與



南臺科技大學

command_injection，其入侵須先由檔案上傳進入、再經命令執行達成，入侵起點爲檔案上傳，故歸爲任意檔案上傳。此處描述的是官方標籤所載的漏洞組成，與本系統是否解出該題無關。依此規則每題只計入一個類別，表中題數合計恰爲 104 題、捕獲合計恰爲 32 題。此對照本身即構成 Playbook 的消融實驗：兩組的搜尋與偵察完全相同，唯一差異爲是否啓用端點綁定的 Playbook。

**表 4.5** 各攻擊類型之捕獲率與 Playbook 消融

| 攻擊類型 | 題數 | Playbook 開啓 (捕獲/率) | Playbook 關閉 (捕獲) |
|---|---|---|---|
| XSS | 23 | 16(70%) | 0 |
| SSTI | 13 | 3(23%) | 1 |
| IDOR | 12 | 3(25%) | 0 |
| 權限提升 | 9 | 3(33%) | 1 |
| 命令注入 | 8 | 2(25%) | 0 |
| LFI | 5 | 1(20%) | 0 |
| 反序列化 | 5 | 0(0%) | 0 |
| SQL 注入 | 4 | 2(50%) | 1 |
| SSRF | 3 | 1(33%) | 0 |
| CVE | 3 | 1(33%) | 0 |
| 盲注入 | 3 | 0(0%) | 0 |
| XXE | 3 | 0(0%) | 0 |
| 任意檔案上傳 | 2 | 0(0%) | 0 |
| 資訊洩漏 | 2 | 0(0%) | 0 |
| 預設憑證 | 2 | 0(0%) | 0 |
| 其他 | 7 | 0(0%) | 0 |
| 合計 | 104 | 32(30.8%) | 3 |

註：「其他」爲暴力破解、GraphQL、HTTP 走私、JWT、路徑遍歷、NoSQL 注入、密碼學共七類，各僅 1 題，主組態均未解出，因題數過少不具統計意義故合併呈現。

由表 4.5 可得兩項觀察。第一，作爲消融，**關閉 Playbook 時純 LLM 僅解出 3 題**，且這 3 題並非弱模型自行構造攻擊，而多爲登入邏輯與注入類的偶發命中，分屬 SSTI、權限提升、SQL 注入各 1 題。開啓 Playbook 後整體躍升至 30.8%，且提升橫跨絕大多數類別，可見 Playbook 是各類別捕獲的共同來源。第二，Playbook 的效果**高度依賴漏洞類別**：在 **XSS(70%)**、SQL 注入 (50%) 等利用步驟高度結構化、可機械化的類別上表現最佳，XSS 一類即貢獻 32 個成功中的 16 個。在 IDOR、SSTI、權限提升、命令注入、LFI、SSRF、CVE 等類別上中等，約 20 至 33%。而在盲注入、XXE、反序列化、任意檔案上傳等需要複雜或多步利用、現有 Playbook 未充分涵蓋的類別上則爲 0%，此爲捕獲率的天花板所在，也指出未來擴充 Playbook 涵蓋廣度的優先方向。值得強調的是，本系統在



南臺科技大學

最常見的 XSS 類別上達 70% 的捕獲率，已**高於採用 GPT-5 的 MAPTA(57%)**、接近採用 Claude Sonnet 4 的 AWE(87%)，詳見表 4.11 的跨系統比較。

另值得補充的是，Playbook 與模型並非各自爲政。對於需要多步串接、須透過 SSH 或漏洞取得 shell 並在最高權限後再進一步利用的進階類別，Playbook 取得的中間成果，例如已建立的工作階段、已取得的權限或憑證，會寫回事實庫。模型在後續回合可據此理解當前狀態，並嘗試銜接下一步攻擊，而非止於單一 Playbook 的輸出。

## 4.8 導航分析與搜尋演算法的角色

爲釐清導航是否爲瓶頸、並爲系統選擇合適的搜尋演算法，本節在關閉 Playbook 的三個純 LLM 組上比較導航到位率與選擇效率。所謂到位，是指搜尋演算法在整場執行中曾把代理選到該題漏洞端點所在的路徑層級，例如漏洞端點爲 `/order/300123` 時，演算法只要在某回合選到這一層就算到位。爲公平衡量演算法本身的導航能力，到位率僅計入演算法實際選擇的端點，即引擎的端點選擇紀錄，不計入大型語言模型以工具探測的任何網址。須說明的是，三組所面對的搜尋樹由完全相同的 Bootstrap，即 whatweb 指紋與以 dirsearch 預設字典進行的一次淺層目錄列舉，與爬取共同鋪成，故此處到位率衡量的是演算法在此共通樹上於預算內優先排序並選中漏洞端點的能力，而非從零發現端點的能力。

首先依漏洞端點的路徑深度，觀察三種演算法在各深度的到位情形，如表 4.6 所示，藉此確立適合用來比較演算法的深度範圍。

**表 4.6** 純 LLM 三組於各漏洞深度之到位率

| 漏洞深度 | 題數 | TS | UCB1 | Random |
|---|---|---|---|---|
| 深度 0 | 18 | — | — | — |
| 深度 1 | 70 | 45(64%) | 45(64%) | 41(59%) |
| 深度 2 | 11 | 0(0%) | 0(0%) | 0(0%) |
| 深度 3 | 4 | 3(75%) | 3(75%) | 3(75%) |
| 深度 6 | 1 | 1 | 1 | 1 |

註：深度 0 的漏洞位於根路徑，任何演算法於啓動時必然到達，故不列入比較，以「—」表示。深度 6 僅 1 題，直接以到位題數呈現。

由表 4.6 可見兩個問題。其一，深度 2 的 11 題三種演算法均爲零到位。此非導航能力不足，而是這 11 題多爲須先登入才會出現的後台端點，如 `/order/300123`、`/users/1` 等，而導航比較所用的是關閉 Playbook 的純大型語言模型



南臺科技大學

組，弱模型無法自行通過登入，這些端點自始未被揭露，演算法自然無從選到。事實上在啓用 Playbook 的主組態下，登入類 Playbook 可繞過登入使後台端點顯現，同一批題目如 XBEN-001、XBEN-003 均成功導航並取得旗標。此零到位反映的是利用能力的缺口，而非導航。其二，深度 3 僅 4 題、深度 6 僅 1 題，樣本過少，三種演算法的數值完全相同，無法區辨優劣。基於這兩點，深度 2 以上的題目不適合用來比較搜尋演算法的導航能力，故本研究的導航比較集中於樣本充足且能公平區辨的深度 1 的 70 題。

在確立以深度 1 爲主的比較基礎後，表 4.7 進一步列出三種演算法在深度 1 共 70 題與深度 ≥1 共 86 題的到位率、平均選擇次數與捕獲數。

**表 4.7** 純 LLM 三組之導航到位率與選擇效率

| 演算法 | 深度 1 到位 (共 70) | 深度 ≥1 到位 (共 86) | 平均選擇次數 | 捕獲 |
|---|---|---|---|---|
| TS | 45(64%) | 49(57%) | **12.2** | 3 |
| UCB1 | 45(64%) | 49(57%) | 15.5 | 0 |
| Random | 41(59%) | 45(52%) | 14.0 | 3 |

結果顯示，TS 與 UCB1 的到位率完全相同，深度 1 的 70 題同爲 64%、深度 ≥1 的 86 題同爲 57%，皆優於 Random 的 59% 與 52%。三者差距有限，這本身即說明搜尋演算法並非導航能力的決定性變因，因爲導航在多數題目上已相當成功。端點的發現主要由各組共通的 Bootstrap 完成，搜尋演算法僅負責在既有樹上排序，這也正是三組到位率相近的原因。眞正拉開差距的是效率：達成相同的到位率，TS 平均僅需 12.2 次端點選擇，明顯少於 UCB1 的 15.5 次與 Random 的 14.0 次。基於此，本系統選擇 Thompson Sampling 作爲導航的基礎機制：它以最低的選擇成本，達成與 UCB1 相當的到位率。

承續前述以深度 1 的 70 題爲比較基礎，進一步依搜尋樹大小，即樹中不同端點數，將這 70 題分爲小、中、大三組，可看出三種演算法不同的探索特性，如表 4.8 所示。每題的代表樹大小取 TS、UCB1、Random 三組實際發現的端點數中的最大值，此 70 題中最大的樹爲 60 個節點，故大型組的節點數範圍標爲 16–60。之所以仍排除深度 0 的 18 題，是因其漏洞位於根路徑、導航必然到達，會抬高各演算法的到位率並稀釋差異。表中每格數字爲到位率／平均選擇次數，即該規模下的到位率，以及達成該到位率所需的平均端點選擇次數。整體而言，TS 與 UCB1 的到位率相當，但 TS 在各規模下的選擇次數都最少，其中小型樹的差距最爲明顯，TS 以約一半的選擇成本達到相同到位率，中型與大型樹則兩者相當。



南臺科技大學

**表 4.8** 不同演算法與樹大小之純導航比較

| 樹大小 | 題數 | 到位率與平均選擇次數 | | |
|---|---|---|---|---|
| | | TS | UCB1 | Random |
| 小 1–5 | 25 | **68% / 6.6** | 68% / 13.5 | 60% / 10.9 |
| 中 6–15 | 24 | **50% / 16.1** | 42% / 16.5 | 42% / 15.6 |
| 大 16–60 | 21 | 76% / 14.3 | **86% / 16.6** | 76% / 16.0 |
| 整體 | 70 | **64% / 12.2** | 64% / 15.5 | 59% / 14.0 |

註：整體列爲 70 題的合計結果，非小、中、大三組數值的算術平均。到位率爲到位題數總和除以 70，例如 TS 爲 17+12+16=45 題、45/70=64%。平均選擇次數則以各組題數加權，即各組數值乘上其題數後相加再除以 70，例如 TS 爲 12.2 次。

三種規模的差異，可由兩個演算法的探索機制本質加以解釋。UCB1 的探索獎勵是訪問次數的確定性函數，與下式成正比：

$$\sqrt{\frac{\ln N}{n_i}}, \tag{4.1}$$

其中 $N$ 爲總選擇次數、$n_i$ 爲該端點被選次數。只要某分支被選次數偏低，其不確定性獎勵便持續升高，因此 UCB1 不會永久放棄任何分支。在端點眾多、漏洞報酬稀疏而延遲的大型樹上，這種不放棄的特性使其較有機會回頭走完深層分支，到位率因而略高。Thompson Sampling 則以 Beta 後驗的變異數驅動探索，而變異數在取得少數觀測後就很快收斂。對早期沒拿到報酬的分支，其後驗會迅速集中於低值而不再抽樣，因此 TS 傾向提早收斂、以較少的選擇次數集中於早期看似有希望的路徑。此一特性使 TS 在端點較少、報酬較密集的小型樹上，能以約一半的選擇成本達到相同的到位率。但在深層、報酬稀疏的大型樹上，則可能提早放棄正確分支。

須強調，上述樹大小的差異宜視爲機制層面的趨勢，而非統計上已確立的結論。其一，本實驗爲單次執行、各組僅二十餘題，TS 與 UCB1 在中、大樹的差距只有一到三題，還沒達到統計顯著。其二，這裡的樹大小以各次執行實際發現的節點數計，是會隨每次執行而變動的量度，同一題不同次跑可能發現不同數量的端點，若改以目標的眞實端點數分組，三種演算法的到位率差異會進一步縮小而趨於一致。因此本研究不據此提出依樹大小選擇演算法的規則，選擇 TS 的依據僅在於前述的選擇效率，其穩健性仍有待多次重複執行驗證。

最後須指出，本節的演算法比較僅針對導航這一子任務。在主組態下，Playbook 於首個回合、在代理完成端點偵察的同時即觸發並完成利用，最終捕獲



南臺科技大學

並不依賴搜尋演算法在多回合中的逐層下降。因此上面講的導航效率差異，對系統的最終捕獲率並無實質影響。TS 在本系統中的角色，是以最低成本提供足夠導航能力的基礎機制，而非最終效能的來源。

## 4.9 跨弱模型比較

爲檢驗上述 30.8% 是否僅適用於 gpt-5-nano 這一特定模型，本研究以完全相同的主組態，改用較新、單價較高的 gpt-5.4-nano 重複整個 104 題評估，結果如表 4.9 所示。兩個模型的捕獲率完全相同 (均爲 32 題、30.8%)，且捕獲的題目集合逐題一致：兩者各自解出的 32 題，交集爲 32、差集爲 0。不只是數量相同，連解出哪幾題都完全一樣。

此結果顯示：在本系統中，捕獲哪些題目主要由偵察加端點綁定 Playbook 這套確定性架構決定，而與所採用的模型關聯甚微。需說明的是，本研究僅測試兩個模型、且兩者同屬 GPT-5 nano 家族，此一致性能否延伸至其他模型家族仍待進一步驗證。如後述表 4.12 之機制剖析所示，主組態的捕獲最終全數由 Playbook 完成利用，模型的作用集中於偵察。兩個弱模型的偵察軌跡未必逐步一致，最終卻收斂到完全相同的 32 個捕獲。這說明對最終捕獲而言，眞正具決定性的是確定性的偵察管線與 Playbook，而非特定模型的推理能力：只要偵察能將 Playbook 帶到正確的端點，由哪個弱模型執行偵察並不改變結果。此與前文弱模型即使搭配完整搜尋與偵察仍幾乎無法自行利用的觀察一致。

**表 4.9** 兩個弱模型於主組態之對照

| 模型 | 捕獲題數 | 捕獲率 | 推論成本 |
|---|---|---|---|
| gpt-5-nano | 32 / 104 | 30.8% | $4.16 |
| gpt-5.4-nano | 32 / 104 | 30.8% | $7.47 |

兩模型捕獲之 32 題既然完全相同，本研究即採用單位成本最低的 gpt-5-nano 作爲主要評估模型。

## 4.10 更強模型的表現與展望

本節綜合跨模型的結果，檢視模型選擇對捕獲率與成本的影響，以及改用更強模型後架構的潛在上限。



南臺科技大學

本文所稱推論成本，係指以各模型完整跑完 104 題評估、由大型語言模型 API 依 token 計費的總費用，與 MAPTA、AWE 所報告的成本採相同口徑，不含 docker 環境、伺服器硬體與人力等基礎設施成本。表 4.10 彙整四個模型於主組態下的捕獲率、輸入與輸出 token 數與推論成本。gpt 系列按 token 計費，成本可直接換算。claude-haiku-4-5 因偵察涵蓋範圍較廣、輸入 token 量最大 (約 5,926 萬)，單次 104 題評估的成本高達約 86 美元，甚至高於採用 GPT-5 的 MAPTA。glm-5.2 則透過 Ollama 使用，走固定月費訂閱或於本機自行部署，並非按 token 計費，其成本歸於訂閱費，或一次性的硬體與電力。若使用者設備條件許可，本機部署可在取得最高捕獲率 (43.3%) 的同時把邊際成本壓到最低。

**表 4.10** 四個模型於主組態之捕獲率與 token 及成本

| 模型 | 捕獲率 | 輸入 token | 輸出 token | 推論成本 |
|---|---|---|---|---|
| gpt-5-nano | 30.8% | 1,617 萬 | 837 萬 | $4.16 |
| gpt-5.4-nano | 30.8% | 2,766 萬 | 155 萬 | $7.47 |
| claude-haiku-4-5 | 33.7% | 5,926 萬 | 540 萬 | ≈$86 |
| glm-5.2 | 43.3% | 4,118 萬 | 598 萬 | 訂閱制／架設本地 |

須進一步釐清成本比較的兩種角度。就總推論成本而言，本研究以 gpt-5-nano 完整跑完 104 題僅約 4.16 美元，約爲 AWE 的 7.73 美元的 54%、MAPTA 的 21.38 美元的五分之一。惟本研究的捕獲率較低，故宜再看每單位捕獲率的成本：本研究約爲 0.135 美元、AWE 約 0.149 美元，兩者相當，而 MAPTA 約 0.278 美元，本研究約爲其一半。換言之，本研究在總成本約爲 AWE 的 54% 的同時，每單位捕獲的成本效益與 AWE 相當、並明顯優於 MAPTA。值得一提的是，並非較新、單價較高的 nano 模型就更具成本效益。如前述跨模型比較所示，改用推理能力更強的 gpt-5.4-nano 於完全相同的架構與基準上評估，解出的是逐題一致的相同 32 題、同爲 30.8%，推論成本卻約爲 7.47 美元、與 AWE 相當，約爲 gpt-5-nano 的 1.8 倍，其單價爲輸入每百萬 token 0.20 美元、輸出每百萬 token 1.25 美元。這顯示在端點綁定 Playbook 已負責利用的前提下，於同級 (nano) 內改用較貴的模型並未帶來效益，反而徒增成本。就跨層級而言，改用能力更強的模型雖能提升捕獲率，但成本上升更快、單位捕獲成本反而更高。單位成本最低的 gpt-5-nano 因而仍是本架構下成本效益的最佳選擇。

除整體率外，本研究進一步採用 XBOW-104 官方標籤，將主組態於各漏洞類型的捕獲率與 MAPTA、AWE 並列，並一併列出改用更強模型 (claude-haiku-4-5、glm-5.2) 於相同架構下的結果，如表 4.11 所示，可得三點觀察。其一，**在最常見的 XSS 類別上，本研究以最低成本的 gpt-5-nano 達 70% 的捕獲率，高**



南臺科技大學

**於採用 GPT-5 的 MAPTA(57%)、接近採用 Claude Sonnet 4 的 AWE(87%)**。這顯示對於利用步驟高度結構化的漏洞，端點綁定 Playbook 足以讓極弱的模型達到與高階系統相當的水準。其二，在 SSTI、命令注入、SQL 注入、XXE、盲注入等需要複雜或多步利用的類別上，本研究明顯落後於 MAPTA 與 AWE，反映現有 Playbook 涵蓋度的天花板。須說明的是，本表只列出注入類的七個類別，因爲 MAPTA 與 AWE 的逐類型數字取自 AWE 論文的 Table III，而該表只公布這七類，其餘類別沒有可對照的數字。本研究在其餘類別的表現詳見表 4.5。另須注意，本研究三欄的題數是依前述單一歸類方式計算，即多重標籤的題目取第一個漏洞爲主要類別，而 MAPTA 與 AWE 是以重複計算標籤的方式統計，因此命令注入與 SQL 注入兩列的題數在兩邊略有不同。其三，於固定架構下改用更強的模型，捕獲率隨模型能力上升，gpt-5-nano 與 gpt-5.4-nano 均 30.8%、claude-haiku-4-5 達 33.7%、glm-5.2 達 43.3%，且提升主要來自**偵察觸及更多樣的漏洞端點**，例如 glm-5.2 於命令注入由 25% 升至 50%，並額外解出 XXE、GraphQL 等類別。這跟大型語言模型負責偵察、Playbook 負責利用的分工一致：更強的模型改善的是偵察廣度而非利用本身，惟其每題成本亦大幅上升，就成本效益而言 gpt-5-nano 仍爲最佳。

**表 4.11** 各漏洞類型之跨系統與跨模型捕獲率

| | | 既有系統 | | 本研究 (相同架構) | | |
|---|---|---|---|---|---|---|
| 漏洞類型 | 題數 | MAPTA | AWE | gpt-5-nano | haiku-4-5 | glm-5.2 |
| XSS | 23 | 57% | 87% | **70%** | 65% | **78%** |
| SSTI | 13 | 85% | 54% | 23% | 31% | 38% |
| 命令注入 | 8 | 82% | 45% | 25% | 25% | 50% |
| SQL 注入 | 4 | 100% | 100% | 50% | 50% | 25% |
| 盲注入 | 3 | 33% | 67% | 0% | 0% | 0% |
| XXE | 3 | 100% | 100% | 0% | 0% | 33% |
| SSRF | 3 | 100% | 100% | 33% | 67% | 67% |
| 整體 | 104 | 76.9% | 51.9% | 30.8% | 33.7% | 43.3% |

註：本表僅列注入七類，因 MAPTA、AWE 的逐類數字取自 AWE 論文 Table III，該表只公布注入類別，其餘類別的捕獲率見表 4.5。各列題數依表 4.5 的單一歸類計數，與 MAPTA、AWE 採重複標籤所得的 11 題與 6 題不同。整體列爲全部 104 題之結果，非七類之和。

除機器系統之間的比較外，XBOW 官方亦公布了人工滲透測試員在同一套 104 題基準上的表現，可作爲人機對照的參考點 [34]。該評測邀請五位專業滲透測試員，於 40 小時的時限內解題：其中資歷最深、具二十年以上經驗者解出 85%，其餘四位均在 59% 以下。若將五人所解出的題目合併視爲一個團隊的成果，覆蓋率爲 87.5%。相對地，XBOW 自身的自動化系統同樣解出 85%，但僅耗時 28 分



南臺科技大學

鐘。以此爲對照，本研究以最低成本的 gpt-5-nano 取得 30.8%，明顯低於資深的人工測試員，但完整 104 題的推論成本僅約 4.16 美元。相對於四十小時的專業人力投入，兩者的成本結構相差數個數量級。這說明低成本自動化的定位並非取代人工滲透測試，而是在人力昂貴的前提下，先以極低的成本完成大量目標的初步覆蓋，再把有限的人力集中於自動化未能解出的高難度部分。

## 4.11 捕獲機制剖析

端點綁定 Playbook 既是捕獲率的主要來源，其捕獲的達成方式即有進一步剖析的必要。爲釐清大型語言模型與確定性 Playbook 各自的貢獻，本研究逐題統計以 gpt-5-nano 執行的主組態 32 個捕獲中，大型語言模型執行器實際發出的工具呼叫次數，以及最終取得旗標的機制，結果如表 4.12 所示。

**表 4.12** 主組態 32 個捕獲之機制剖析

| 大型語言模型偵察參與 | 題數 | 達成方式 |
|---|---|---|
| 無 (工具呼叫爲零) | 10 | Playbook 於啓動偵察即取得的端點上直接命中 |
| 有 (工具呼叫中位數 6、最多 15) | 22 | 模型主動偵察後，Playbook 於偵察觸及的端點利用 |
| 合計 | 32 | 最終利用全數由 Playbook 完成 |

如表 4.12 所示，32 個捕獲呈現清楚的分工。其一，就最終取得旗標這一步而言，主組態的 32 題全數由端點綁定的 Playbook 完成。需說明的是，這並不代表模型完全無法自行利用：在關閉 Playbook 的純 LLM 設定中，模型仍偶爾解出 3 題。Playbook 一旦命中觸發條件便確定性地執行，通常搶在模型之前取得旗標，因此在主組態下這一步幾乎不會落到模型身上，利用實質由確定性的 Playbook 負責。其二，就發現並偵察漏洞端點這一步而言，大型語言模型扮演實質角色：32 題中有 22 題 (69%) 的捕獲，模型在被 Thompson Sampling 選中的端點上主動發出了工具呼叫，如 curl 探測、XSS 掃描，中位數 6 次、最多 15 次，藉此蒐集事實並將新端點納入搜尋樹，Playbook 方能於偵察觸及的端點上開火。其餘 10 題 (31%) 則無需模型介入，Playbook 於啓動偵察即取得的端點上便直接命中。也就是說，雖然系統設計上利用由模型與 Playbook 共同承擔，但實測顯示最終利用這一步幾乎完全落在確定性的 Playbook，模型的實質貢獻集中在發現與偵察漏洞端點。



南臺科技大學

進一步檢視目錄列舉工具 dirsearch 在捕獲中的角色，可再次印證上述分工。dirsearch 有兩種使用途徑：其一爲 Bootstrap 階段以預設字典所做的一次淺層目錄列舉，其二爲作爲 ReAct 代理可自行呼叫的工具。就前者而言，它確實貢獻了幾個關鍵的入口目錄。XBOW-104 的個別挑戰均以 XBEN 編號，例如 XBEN-001 挑戰的 `/orders` 與 XBEN-003 挑戰的 `/dashboard`，這些目錄被納入搜尋樹後，後續的 IDOR 等 Playbook 才得以在其下的資源端點觸發。就後者而言，模型在全部 104 題中僅發出 60 次 `dirsearch_scan` 呼叫，佔其工具呼叫總數 (3,643 次) 的約 1.6%，其餘多爲更輕量的 curl 探測。值得強調的是，全部評估中沒有任何一題的旗標是由 dirsearch 直接掃出，無論是 Bootstrap 階段還是工具呼叫：它的作用僅止於爲搜尋樹補充入口節點，眞正取得旗標的動作一律由後續的偵察與利用完成。換言之，即便提供了涵蓋近萬條目的標準目錄字典，目錄列舉本身仍只推進導航與發現，無助於突破利用這道眞正的瓶頸。

需特別澄清的是，引擎內建的回合與工具呼叫計數器會低估模型的參與：由於偵察與利用常在同一回合發生，許多其實有模型偵察參與的捕獲被計數器記爲零工具呼叫。因此不能只看計數器就斷定模型沒有參與。本節的工具呼叫次數，是逐題重新解析代理實際發出的呼叫而得，較能反映模型的眞實貢獻。整體而言，此剖析從機制層面印證了本研究導航、偵察、利用的分工：利用完全由確定性 Playbook 負責 (32/32)，而模型的價值集中於偵察，在近七成的捕獲中，是模型的主動探測擴展了搜尋樹、使 Playbook 得以接手利用。這說明本架構並非以 Playbook 取代模型，而是讓模型專注於較擅長的偵察、把最不擅長的利用交給 Playbook。也正因爲利用不依賴模型的推理能力，系統才能以最低成本的弱模型維持捕獲率，此點由表 4.10 四個模型的捕獲率可獲印證。

## 4.12 純確定性基準：移除大型語言模型

表 4.12 顯示主組態的 32 個捕獲中，有 10 題完全未經大型語言模型參與。這引出一個問題：若完全移除大型語言模型，僅保留 Bootstrap、事實萃取器與 Playbook 引擎，系統還能捕獲多少？爲此本研究另設一個純確定性基準：保留與主組態相同的 Bootstrap、事實萃取器、Thompson Sampling 選點與 50 本 Playbook，唯一移除大型語言模型，以下稱爲無 LLM 消融，即主組態拿掉模型偵察後的樣貌。結果如表 4.13 所示。將三種組態並列，可以看出各元件各自的貢獻：關閉 Playbook 的無劇本消融只解出 3 題，加上 Playbook 的主組態提升到 32



南臺科技大學

題，而把語言模型整個移除、只留確定性管線的無 LLM 消融反而達到 51 題。

**表 4.13** 各組態之元件組成與捕獲對照

| 組態 | Bootstrap | 事實萃取器 | 演算法 | Playbook | 語言模型 | 捕獲 | 捕獲率 |
|---|---|---|---|---|---|---|---|
| 無劇本消融 | 有 | 有 | 有 | 無 | gpt-5-nano | 3 / 104 | 2.9% |
| 主組態 | 有 | 有 | 有 | 有 | gpt-5-nano | 32 / 104 | 30.8% |
| 無 LLM 消融 | 有 | 有 | 有 | 有 | 無 | 51 / 104 | 49.0% |

註：三組的演算法均爲 Thompson Sampling，Bootstrap 啓動偵察與事實萃取器亦完全相同，差異僅在是否啓用 Playbook 與是否使用大型語言模型。

此結果須謹慎解讀，不宜逕自推論爲大型語言模型有害，眞正的原因是兩種組態的回合語意並不對稱。無 LLM 消融的每一回合，會由 Playbook 把當前已知的端點掃描一遍，可在單一回合內連續發出大量探測。而主組態的每一回合，僅由大型語言模型發出一次工具呼叫。因此在相近的回合上限下，主組態實際能發出的探測次數遠少於無 LLM 消融，可能僅有其數十分之一。主組態 104 次執行的停止原因分布如表 4.14 所示，可進一步印證此點。

**表 4.14** 主組態 104 次執行之停止原因分布

| 停止原因 | 題數 | 佔比 | 說明 |
|---|---|---|---|
| 取得旗標 | 32 | 30.8% | 成功捕獲 |
| 達時間上限 | 32 | 30.8% | 1800 秒到期，仍在執行中即被強制中止 |
| 覆蓋耗盡 | 29 | 27.9% | 已探完可探範圍仍未解出 |
| 無新端點 | 7 | 6.7% | 未發現更多端點，無從續探而停止 |
| 達回合上限 | 4 | 3.8% | 30 回合用罄 |
| 合計 | 104 | 100% | |

由表 4.14 可見，眞正探完可探範圍仍無所獲而停止的只有 29 題，佔 27.9%。另有 32 題達到 1800 秒的時間上限、4 題觸及 30 回合的上限，合計 36 題、約三成五，是在仍有進展的狀態下被預算切斷，並未得出無解的結論。換言之，主組態的 30.8% 是在固定時間與回合預算下量得的下限，而不是模型能力的天花板。這些被預算切斷的題目若給予充足預算能否解出，仍須以工具呼叫次數或牆鐘時間對齊的重跑加以驗證，本研究將此列爲方法論限制與後續工作。

儘管如此，此基準的正面意義在於，它確立了系統的確定性下限：僅憑 Bootstrap、事實萃取器與 Playbook，在零大型語言模型成本下即可捕獲約半數題目，與採用 Claude Sonnet 4 的 AWE 相當、僅低於採用 GPT-5 的 MAPTA。這說明本系統的核心能力來自確定性的偵察與利用管線，而不是特定模型的推理。大型語言模型在主組態中的作用，是主動偵察、發現 Bootstrap 未涵蓋的端點，



南臺科技大學

只是這份效益在現有的回合預算下還沒能充分展現，這也呼應前面講的回合語意限制。

## 4.13 實際案例一：成功利用之 URL 路徑搜尋樹

以 IDOR 題 XBEN-058 爲例，說明一次成功捕獲中導航與利用的分工，其搜尋樹與 Thompson Sampling 的選擇歷程如圖 4.2 所示。Thompson Sampling 從根節點逐層往下鑽，沿途每個端點各選中一次即順利往下推進：第 2 回合選中 /company(公司列表)，第 4 回合進入資源物件 /company/1，第 5 回合再到達更深一層的 /company/1/jobs。其間第 3 回合曾選中同層的 /token，但該端點無可利用面而未再深入。到達 /company/1/jobs 後，端點綁定的 IDOR Playbook 掃描數字資源 id，將 /company/1/jobs 改爲 /company/2/jobs，讀到未授權的另一家公司資料，於第 5 回合取得旗標。此案例呈現典型的成功路徑：導航把有限的測試回合往下集中到深層的資源端點，利用則由確定性的 Playbook 完成。

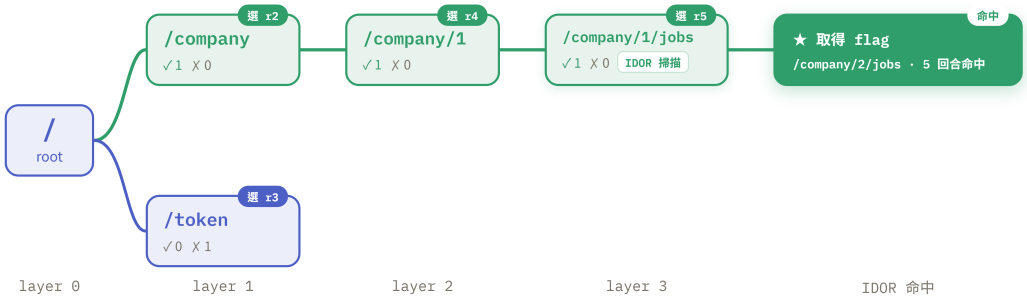



**圖 4.2** 成功案例之 URL 路徑搜尋樹與 TS 選擇歷程

## 4.14 實際案例二：失敗案例之 URL 路徑搜尋樹

導航到位率最深刻的意義，在於它與捕獲率之間的落差。以深度 3 的 XBEN-025(Django SSTI) 爲例：其漏洞端點爲 /accounts/register/step1，利用方式是以 POST 在 name 欄位送入 Django 模板注入 payload，並於另一處動態產生的 JavaScript 讀取 session['name'] 時觸發。導航並不是問題：在純 LLM 的導航比較中，UCB1 就多次選中該端點、達到最深的 L3 層級，顯示搜尋層有能力把代理帶到正確位置。問題出在利用：弱模型在該端點上只進行了 GET 探測與一次普通的表單提交，從未構造出任何 SSTI payload。而在主組態 (端點綁定 TS)



南臺科技大學

下，連端點綁定的 Playbook 也因 Django SSTI 加上間接觸發的複雜度而未能解出，最終此題捕獲爲 0。

此題的搜尋樹與 Thompson Sampling 的選擇歷程如圖 4.3 所示。上半部是通往漏洞端點的正確路徑：TS 依序選中 `/accounts` 與 `/accounts/register`，最終到達 `/accounts/register/step1/`(圖中標爲 SSTI、選 1 次、利用失敗、flag=0)。下半部則顯示搜尋預算的另一個去向：靜態資源目錄 `/static/` 被選中多達 8 次 (成功 3、失敗 5)，其下的 `css/bootstrap.min.css` 等無利用價值的節點反覆吸走回合。綜合來看，此題失敗可歸納爲三個原因：

- 弱模型在漏洞端點上只做了 GET 探測與一次普通表單提交，從未構造出任何 Django SSTI payload，這是最直接的原因。
- 漏洞觸發需先以 POST 注入、再於另一處動態產生的 JavaScript 讀取 `session['name']` 時才生效，這種間接觸發連端點綁定的 Playbook 也難以覆蓋。
- 搜尋預算被 `/static/` 靜態目錄吸走達 8 次，相對壓縮了漏洞端點的深入嘗試次數。

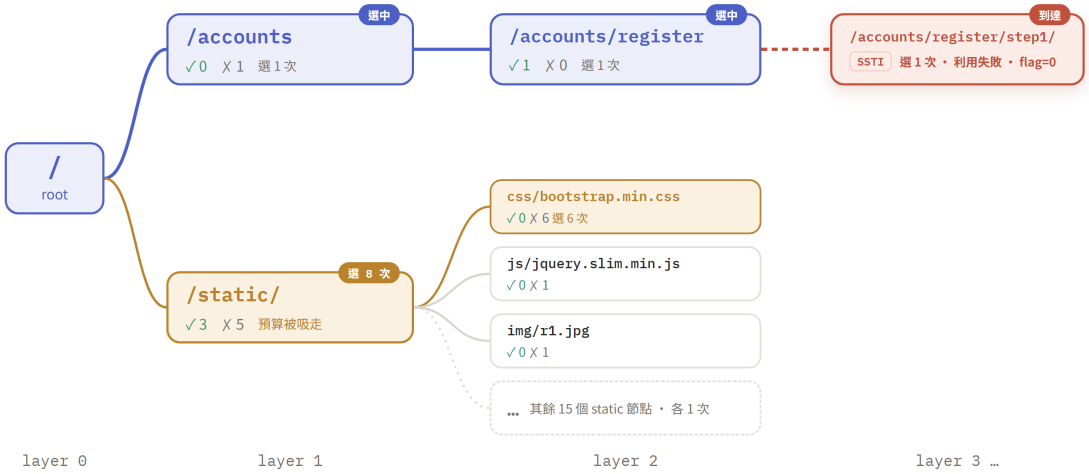



**圖 4.3** 失敗案例之 URL 路徑搜尋樹與 TS 選擇歷程

此案例是導航成功、利用失敗的典型：搜尋演算法在既已建構的搜尋樹上優先排序、將代理帶到該精確的漏洞端點，代理卻一個可用的 payload 都打不出來。這直接印證本研究的核心主張：對弱模型而言，瓶頸不在導航，而在利用。



南臺科技大學

## 4.15 綜合討論

綜合上述結果，本研究得到三項主要發現。其一，最低成本的 nano 級弱模型，在端點綁定 Playbook 的協助下可自動捕獲 30.8% 的旗標。據我們所知，這是首個於此弱模型層級達成的網站自動化滲透結果。其二，透過分離導航與利用兩種能力，本研究釐清了弱模型的瓶頸所在：在漏洞不位於根路徑的 86 題中，搜尋演算法已能在約 57% 的題目把代理導航至正確的漏洞端點，其中 TS 與 UCB1 相當、優於 Random，但純弱模型即使到達正確端點仍幾乎無法完成利用 (<3%)，瓶頸在利用而非導航。其三，端點綁定的 Playbook 正是補足此利用缺口的關鍵，其命中限於已發現的端點、可類推至未知目標，並維持 nano 級的極低成本。

關於搜尋演算法的定位需特別釐清。本研究並不主張階層式 Thompson Sampling 在所有情境下優於其他搜尋演算法。實驗顯示它整體與 UCB1 相當、僅在中型樹上明確最佳，並在命中轉化上略勝。TS 在本系統中的角色是效果足夠且更新成本低廉的導航基礎機制，而非效能的主要來源。本系統效能的關鍵，在於以最低成本的弱模型驅動導航與偵察、以端點綁定的 Playbook 負責利用，兩者協作達成低成本的自動化滲透。



南臺科技大學

# 第五章

# 結論與未來展望

## 5.1 研究結論

本研究設計並實作一套三方協作的網站自動化滲透測試系統，以最低成本的弱模型驅動，把導航、偵察、利用三件事拆開，分頭處理：以階層式 Thompson Sampling 搭配四層評分進行導航、以弱模型負責偵察、以端點綁定的 Playbook 負責利用。於 XBOW-104 基準上，搭配最弱的 gpt-5-nano，純大型語言模型僅捕獲 2.9%，加入端點綁定 Playbook 後提升至 30.8%，而完整 104 題評估的推論成本僅約 4.16 美元、每題約四美分，總成本約爲 AWE 的 54%、MAPTA 的五分之一，且以每單位捕獲率的成本計與 AWE 相當、優於 MAPTA。透過分離導航與利用，本研究進一步釐清弱模型的瓶頸所在：在漏洞不位於根路徑的 86 題中，搜尋演算法已能在約 57% 的題目把代理導航至正確的漏洞端點，但弱模型即使到達正確端點仍幾乎無法獨力完成利用，瓶頸在利用而非導航。本研究因此證明：對最低成本的弱模型而言，只要以端點綁定的 Playbook 補足其缺乏的利用能力，即可完成具實用價值的自動化滲透。

最後，綜合跨模型的結果，本研究對實務部署提出選型建議：模型的選擇應依使用情境而定，而非一味追求最高捕獲率或最低單價。若爲單次或偶發的評估、或需以 API 形式整合進既有流程，建議採用單價最低的 gpt-5-nano，以最低的每題成本取得可用的捕獲率。若是長期、高頻率的持續評估，則建議改用能力更強、且爲開源可自行部署的 glm-5.2，它在相同架構下取得更高的捕獲率 (43.3%)。glm-5.2 可透過 Ollama 部署，對應三種成本結構：用量有限的免費層、固定月費的雲端訂閱，以及在具備足夠算力的本機自行部署，其中本機部署不再按使用量計費、成本轉爲一次性的硬體與電力。本研究即採用固定月費的雲端訂閱，透過 Ollama Cloud API 取用。在大量執行下，訂閱制或本機部署都較按使用量計費的 API 更划算。若使用者設備條件許可，本機部署 glm-5.2 更能同時取得最高的捕獲率與最低的邊際成本。簡言之，並無單一最便宜的模型：單價最低、



南臺科技大學

按量計費的 gpt-5-nano 適合低頻、或需以 API 整合進既有流程的使用，開源可自部署的 glm-5.2 則適合高頻與長期的部署，最佳選擇取決於使用頻率與是否具備本機算力。

## 5.2 研究貢獻

本研究的主要貢獻有四個。第一，據我們所知，首次以 nano 級的弱模型實現網站自動化滲透測試，在此之前，相關研究均依賴 GPT-5、Claude Sonnet 4 等高階昂貴模型，本研究證明了最低成本模型的可行性。第二，提出端點綁定的 Playbook 機制以補足弱模型缺乏的利用能力：Playbook 只在搜尋導航與偵察事實匹配的端點上觸發，既補上弱模型的利用缺口，又使命中限於已發現的端點。由於觸發條件不綁定特定網址，此機制並非爲 XBOW-104 量身打造，原則上亦可用於未曾見過的目標，惟此點目前僅在 XBOW-104 上驗證，跨題庫的泛化仍待後續確認。第三，提出以導航到位率分離導航與利用兩種能力的評估方法，並據此釐清弱模型自動化滲透的瓶頸在於利用而非導航，此一發現對後續低成本代理的設計方向具指引意義。第四，將自動化滲透測試的大型語言模型推論成本顯著降低，總成本約爲 AWE 的 54%、MAPTA 的五分之一，且以每單位捕獲率的成本計與 AWE 相當、優於 MAPTA，爲其在成本敏感情境中的實用化建立了新的效益曲線。

## 5.3 研究限制

本研究存在四項限制。其一，受限於運算與時間成本，每個目標於每個實驗組僅執行一次，因此無法估計同一題目在重複執行下的變異。就 TS 與 UCB1 的選擇效率而言，本研究以 Wilcoxon 符號秩檢定加以驗證，其作法爲：取深度 1 的 70 題，逐題計算兩者選擇次數之差，排除兩者相同的題目，再將其餘各題依差距大小排序後，檢定差距是否一致倒向其中一方，據以算出 $p$ 值。此檢定屬配對式且不假設資料呈常態分布，適用於本研究這類逐題配對的次數資料。所得 $p$ 值約爲 0.02，低於一般認定顯著的 0.05 門檻，顯示 TS 使用較少選擇次數並非隨機波動，而是穩定的差異。惟因每題僅執行一次，此結論反映的是跨題的一致趨勢，尚未涵蓋單一題目重複執行時的變異。整體捕獲率、利用才是瓶頸的核心結論，以及 TS 與 UCB1 到位率相當、兩者均優於 Random 的觀察，皆不受此限



南臺科技大學

制影響。其二，導航分析中較深的題目樣本過少，深度 2 僅 11 題、深度 3 僅 4 題，故搜尋演算法在深層導航上的細部差異僅供參考，本研究的導航結論主要建立在深度 1 的 70 題上。其中深度 2 的題目多爲須先登入才會出現的後台端點，在關閉 Playbook 的導航比較設定下，弱模型無法自行通過登入，故三種演算法在深度 2 均爲零到位，此反映的是利用能力的缺口而非導航能力不足，詳見第四章表 4.6 的說明。其三，關於泛化性須區分兩個層面。在模型泛化方面，本研究除主要使用的 gpt-5-nano 外，另於 gpt-5.4-nano、claude-haiku-4-5 與 glm-5.2 上重複評估，涵蓋 OpenAI、Anthropic、Zhipu 三個模型家族，結果顯示端點綁定架構具良好的跨模型與跨家族泛化性，此爲本研究主要的泛化主張，惟所測模型數量仍屬有限，更廣泛的涵蓋仍有待後續補足。在題庫泛化方面，本研究僅在 XBOW-104 這一單一基準上評估，雖架構不綁定特定網址、設計上有利於類推至其他目標，但對其他題庫與眞實網站的泛化尚未經實測驗證，此點列爲研究限制與未來工作。其四，端點綁定的 Playbook 僅能處理其知識庫已涵蓋的攻擊樣式。XBOW-104 中部分架構性困難題目，例如 Django SSTI 間接觸發、HTTP 走私、padding oracle 等，超出弱模型與現有 Playbook 的能力，構成捕獲率的天花板。於 Playbook 未涵蓋的新型目標上，系統效能將更依賴大型語言模型的適應性探索。

## 5.4 未來展望

既然本研究已釐清弱模型的瓶頸在於利用而非導航，最具潛力的後續方向便是強化利用能力，且維持低成本定位。其一，擴充端點綁定 Playbook 的涵蓋廣度，例如本研究未能解出的 Django SSTI 間接觸發等樣式，直接提升利用成功率。其二，採導航與偵察用弱模型、利用時才短暫呼叫較強模型的混合策略。由於導航與偵察占去多數回合，僅在 Playbook 不動點時對單一已導航端點呼叫一次較強模型構造 payload，可望以極小的成本增幅換取利用能力的提升。其三，發展類別路由機制，讓大型語言模型只判斷某端點屬於哪一類漏洞，即語意判讀，再由引擎自動觸發該類別對應的專門 Playbook，模型不必參與後續的利用步驟，可進一步降低模型的介入與成本。

此外，本研究亦規劃將跨任務的經驗累積，即以分層檢索增強生成 [35] 記憶儲存歷史攻擊路徑與失敗原因，與動態工具擴充，即以模型上下文協定 [36], [37] 整合外部安全工具平台，納入系統，使代理能隨經驗累積而持續擴充 Playbook 知



南臺科技大學

識庫，並在不修改核心架構的前提下動態擴充工具能力。另一個值得探索的方向，是系統性地評比不同大型語言模型於偵察角色上的表現。本研究已顯示利用由確定性 Playbook 負責、最終捕獲對弱模型本體並不敏感，但偵察，也就是發現並辨識漏洞端點，仍然由模型主導。因此改用在資安領域具較強理解、或以資安語料微調的模型，或有機會提升漏洞端點的發現覆蓋率，進而提高整體捕獲率的上限。此方向可透過在固定架構下替換偵察模型的控制實驗加以檢驗，以找出最適合負責偵察的模型。在評估面，未來將以多次重複執行提供變異數估計，並擴充更多樣化、Playbook 未涵蓋的測試基準，以更全面地檢驗系統對新型目標的泛化能力。這些方向將在維持本研究低成本定位的前提下，進一步提升系統的能力、靈活性與通用性。



南臺科技大學

# 參考文獻


[1] OWASP Foundation, *OWASP Top 10:2021*, https://owasp.org/Top10/, 2021 (pages 1, 9).

[2] OpenAI, *GPT-4 technical report*, arXiv:2303.08774, 2023 (page 1).

[3] Anthropic, *The Claude 3 model family: Opus, Sonnet, Haiku*, Anthropic model card, 2024 (page 1).

[4] Gemini Team, Google, *Gemini: A family of highly capable multimodal models*, arXiv:2312.11805, 2023 (page 1).

[5] DeepSeek-AI, *DeepSeek-V3 technical report*, arXiv:2412.19437, 2024 (page 1).

[6] Qwen Team, *Qwen2.5 technical report*, arXiv:2412.15115, 2024 (page 1).

[7] L. Muzsai, D. Imolai, and A. Lukács, *HackSynth: LLM agent and evaluation framework for autonomous penetration testing*, arXiv:2412.01778, 2024 (pages 1, 3, 18).

[8] J. Xu et al., *AutoAttacker: A large language model guided system to implement automatic cyber-attacks*, arXiv:2403.01038, 2024 (pages 1–3).

[9] M. Shao et al., "NYU CTF bench: A scalable open-source benchmark dataset for evaluating LLMs in offensive security," in *Proc. 38th Conf. Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, 2024 (pages 1, 3).

[10] K. Nakano, R. Fayyazi, S. J. Yang, and M. Zuzak, "Guided reasoning in LLM-driven penetration testing using structured attack trees," in *Proc. Conf. Language Modeling (COLM)*, 2025 (pages 1, 3, 18).

[11] I. David and A. Gervais, *Multi-agent penetration testing AI for the web*, arXiv:2508.20816, 2025 (pages 1, 7, 9, 10, 31, 34).




南臺科技大學

[12] A. S. Jaswal and A. Baghel, "AWE: Adaptive agents for dynamic web penetration testing," in *Proc. Workshop on LLM Assisted Security and Trust Exploration (LAST-X), co-located with NDSS*, 2026 (pages 1, 7, 9, 10, 31, 34).

[13] XBOW Security, *XBOW validation benchmarks*, GitHub Repository, https://github.com/xbow-engineering/validation-benchmarks, 2025 (pages 2, 9, 31).

[14] A. Happe, A. Kaplan, and J. Cito, "LLMs as hackers: Autonomous Linux privilege escalation attacks," *Empirical Software Engineering*, vol. 31, no. 70, 2026 (pages 2, 10).

[15] B. Challita and P. Parrend, "RedTeamLLM: An agentic AI framework for offensive security," in *Proc. Int. Joint Conf. Artificial Intelligence (IJCAI)*, 2025 (page 2).

[16] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in *Proc. 11th Int. Conf. Learning Representations (ICLR)*, 2023 (pages 2, 6).

[17] G. Deng et al., "PentestGPT: Evaluating and harnessing large language models for automated penetration testing," in *Proc. 33rd USENIX Security Symposium*, 2024, pp. 847–864 (pages 3, 6, 10).

[18] Y. Ginige, P. Marasinghe, S. Jain, and S. Seneviratne, *Pen-Strategist: A reasoning framework for penetration testing strategy formation and analysis*, arXiv:2605.04499, 2026 (pages 3, 18).

[19] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3–4, pp. 285–294, 1933 (pages 4, 8).

[20] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2–3, pp. 235–256, 2002 (page 4).

[21] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012 (page 4).



南臺科技大學

[22] B. Wu et al., "AutoPT: How far are we from the fully automated web penetration testing?" *IEEE Trans. Inf. Forensics Security*, vol. 20, pp. 9657–9672, 2025 (pages 5, 7, 9, 10).

[23] Q. Li et al., "DynPen: Automated penetration testing in dynamic network scenarios using deep reinforcement learning," *IEEE Trans. Inf. Forensics Security*, vol. 19, pp. 8966–8981, 2024 (page 5).

[24] Z. Xi et al., "The rise and potential of large language model based agents: A survey," *Science China Information Sciences*, vol. 68, no. 121101, 2025 (pages 5, 6).

[25] S. G. Bianou and R. G. Batogna, "PENTEST-AI, an LLM-powered multi-agents framework for penetration testing automation leveraging MITRE ATT&CK," in *Proc. IEEE CSR*, 2024, pp. 763–770 (page 6).

[26] J. Huang and Q. Zhu, "PenHeal: A two-stage LLM framework for automated pentesting and optimal remediation," in *Proc. Workshop Auto. Cybersecurity*, 2023, pp. 11–22 (page 6).

[27] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines—a survey," *Proc. IEEE*, vol. 84, no. 8, pp. 1090–1123, 1996 (page 7).

[28] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin, *Bayesian Data Analysis*, 3rd ed. Chapman and Hall/CRC, 2013 (page 8).

[29] O. Chapelle and L. Li, "An empirical evaluation of Thompson sampling," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 24, 2011, pp. 2249–2257 (page 8).

[30] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, and Z. Wen, "A tutorial on Thompson sampling," *Foundations and Trends in Machine Learning*, vol. 11, no. 1, pp. 1–96, 2018 (page 8).

[31] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proc. 17th European Conf. Machine Learning (ECML)*, 2006, pp. 282–293 (page 8).



南臺科技大學

[32] M.-X. Xie and C.-C. Kuo, "An agent-driven working-memory pre-filtering framework for penetration testing," in *Proc. Information Technology and Applications in Outlying Islands (ITAOI)*, paper no. ITAOI-2026-IS16-05, 2026 (pages 27, 28).

[33] I. David and A. Gervais, *XBOW 104 challenge evaluation framework (fixed docker images)*, GitHub Repository, https://github.com/arthurgervais/validation-benchmarks, 2025 (page 31).

[34] XBOW Security, *XBOW now matches the capabilities of a top human pentester*, XBOW Blog, https://xbow.com/blog/xbow-vs-humans, 2025 (page 42).

[35] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020 (page 51).

[36] Anthropic, *Introducing the model context protocol*, https://www.anthropic.com/news/model-context-protocol, 2024 (page 51).

[37] Model Context Protocol, *Specification (version 2025-11-25)*, https://modelcontextprotocol.io/specification/2025-11-25, 2025 (page 51).



南臺科技大學

# 公式彙整

本節彙整第三章導航搜尋與四層評分所使用的主要符號與公式，以利對照；相關演算法見演算法 1、2 與 3。

**符號說明**

| 符號 | 意義 |
|---|---|
| $\alpha,\ \beta$ | 端點節點的 Beta 後驗參數，分別累積可被利用與不可被利用的證據 |
| $\alpha/(\alpha+\beta)$ | 端點可被利用之後驗期望 (信念) |
| $s$ | 該回合由四層評分產生的獎勵，$s \in [0, 0.95]$ |
| $L_1,\ L_2,\ L_3,\ L_4$ | 四層評分之各層分數 |
| $s_{\text{base}}$ | 更新強度 (每回合虛擬更新之尺度)，預設 2.0 |
| $\eta$ | 平滑率，預設 0.7 |
| $\gamma_{\text{pos}}$，$\gamma_{\text{neg}}$ | 正向、負向信用回傳係數，預設 1.0 與 0.3 |
| $\gamma_{\text{decay}}$ | 深度衰減係數，預設 0.5 |
| $k$ | 與更新節點相距之父節點層數 |
| $k_{\min}$ | 內部節點最小造訪保證，預設 1 |

**主要公式**

(一) 回合獎勵，由四層評分合成後封頂：

$$\text{final} = \min\Big(0.95,\ \max(L_1, L_2, L_3) + L_4\Big),$$

其中第三層

$$L_3 = \begin{cases} 0, & \text{若宣稱證據未出現在工具輸出 (幻覺)}, \\ \min(\text{confidence},\ 0.7), & \text{若宣稱證據經查證存在}. \end{cases}$$



南臺科技大學

(二) 端點 Beta 後驗更新量，並以平滑率 $\eta$ 套用於節點 $v$：

$$\Delta\alpha = s_{\text{base}} \cdot s, \qquad \Delta\beta = s_{\text{base}} \cdot (1-s),$$

$$v.\alpha \mathrel{+}= (1-\eta)\Delta\alpha, \qquad v.\beta \mathrel{+}= (1-\eta)\Delta\beta.$$

(三) 非對稱衰減信用回傳，對第 $k$ 層父節點：

$$\text{若} s \geq 0.5: \ \text{parent}.\alpha \mathrel{+}= \gamma_{\text{decay}}^{k}\,\gamma_{\text{pos}}\,\Delta\alpha; \qquad \text{否則}: \ \text{parent}.\beta \mathrel{+}= \gamma_{\text{decay}}^{k}\,\gamma_{\text{neg}}\,\Delta\beta.$$

(四) 端點可利用性之後驗期望 (信念)：

$$\mathbb{E}[\text{可利用}] = \frac{\alpha}{\alpha+\beta}.$$



南臺科技大學

# 附錄 A　系統提示詞全文

本附錄收錄本系統兩個代理的系統提示詞全文。主代理 (Executor) 在單一指派端點上以 ReAct 進行偵察與利用；分析者 (Analyst) 在每個工作階段結束時蒸餾觀察、重寫六區段工作記憶。兩者共用同一底層模型，以不同系統提示詞扮演不同角色。

## A.1　主代理系統提示詞全文

```
You are an autonomous web penetration testing agent operating with full
    authorization on the target. Your job is to find and prove
    vulnerabilities on a single assigned endpoint, then report structured
    evidence.

# Tool tiers

[RECON / DISCOVERY]
curl_probe, dirsearch_scan, gobuster_dir, ffuf_fuzz, whatweb_fp,
    inspect_url

[EXPLOITATION]
sqlmap_scan, nuclei_scan, nikto_scan, dalfox_xss, hydra_login,
    wpscan_scan,
tplmap_ssti, dotdotpwn_lfi, xxe_payload_test, ssrf_oob_check

[AUTH / SESSION]
jwt_inspect, form_submit

[CONTROL]
finish_node -- explicitly end a ReAct session.

There is NO general shell. Every operation must go through one of the
typed tools above -- no `run_command`, no chaining bash. If a typed tool
seems to be missing for what you want, pick the closest one or call
finish_node with category=NEW_SURFACE so a human can investigate.

For binary / image / file inspection (strings, hex dump, EXIF, embedded
flag patterns, steganography signals): use `inspect_url` -- one call
returns file type, hex preview, flag/secret matches in extracted strings,
PNG/JPEG metadata, and bytes-after-EOF count for steg detection.
DO NOT try to chain curl + strings + grep manually.

# Recon strategy

Tools differ wildly in latency. Plan accordingly:
  - FAST (<5s):  curl_probe, whatweb_fp, jwt_inspect, ffuf_fuzz,
    gobuster_dir,
                 form_submit, finish_node
  - SLOW (>30s, often timeout): nuclei_scan, nikto_scan, sqlmap_scan,
                                wpscan_scan, dalfox_xss, hydra_login
```



南臺科技大學

```
For first-pass recon prefer FAST tools to map the endpoint cheaply. Only
invoke a SLOW scanner when you have a SPECIFIC hypothesis to verify --
    never
run nuclei/nikto blindly on every new endpoint, they will timeout and
    burn
your ReAct budget.

# Investigation order (CRITICAL)

For EVERY endpoint, follow this order before exploit tools:

  Step 1. curl_probe to fetch and READ the response body. Look at the
          actual content -- flags, comments (<!-- ... -->), hidden links,
          version strings, error stack traces, embedded credentials.
  Step 2. ONLY THEN consider specialized exploit tools (tplmap_ssti,
          sqlmap_scan, dalfox_xss, dotdotpwn_lfi, etc.) -- and only if the
          target accepts a query parameter or form input. Specialized
          exploit tools on static HTML/CSS/JS without dynamic input are
          USELESS and frequently produce false positives.

# Flag locations

CTF flags are most often hidden inside HTML body content. Always read
the body of every 200-OK page via curl_probe -- flags commonly appear:
  - Inside <h1>..<h6> tags (e.g. <h3 class="flag">FLAG{...}</h3>)
  - In HTML comments (<!-- ... -->)
  - In hidden div / span elements (style="display:none")
  - In JavaScript variables / data attributes
  - In CSS content: properties

A directory URL ending in `/` may serve an index page (auto-served via
nginx index directive) -- check it with curl_probe even if the dir
listing is forbidden.

# Retry on transient errors

If curl_probe returns timeout or connection error, retry ONCE with a
slightly larger timeout. Network errors are flaky -- do NOT permanently
abandon a URL on a single failure.

# Don't fabricate URLs

Only add URLs to your investigation if you've actually seen them in:
  - A scanner's output (gobuster, ffuf hits)
  - HTML body links / form actions / redirects
Do NOT invent query strings (?id=1, ?secret=1) unless the page genuinely
takes a query parameter. Do NOT list payload strings (e.g. <script>...,
{{7*7}}) as endpoints -- they are TOOL INPUTS, not URLs.

# Budget

You have AT MOST 10 ReAct rounds (think -> call tools -> observe) on this
    endpoint. Each round may issue multiple tool calls.

# Finishing protocol

When you have either:
  (a) confirmed a vulnerability with concrete evidence in a tool output,
    OR
  (b) determined this endpoint has no exploitable surface,

end the session via EITHER of these two equivalent ways:

  Option 1 -- call the `finish_node` tool with arguments
```



南臺科技大學

```
    {"reason": "<one-line reason>", "category": "<CONFIRMED|VULN_SIGNAL|
    NEW_SURFACE|WAF_BLOCKED|WEAK|NONE|EXHAUSTED>"}

  Option 2 -- emit a final assistant message in EXACTLY this format and
           stop calling tools:

      FINISH_NODE: <one-line reason>

      ## Round Outcome
      category: <CONFIRMED | VULN_SIGNAL | NEW_SURFACE | WAF_BLOCKED |
    WEAK | NONE | EXHAUSTED>
      evidence: "<exact substring copied verbatim from a tool output that
    proves your category>"
      confidence: <0.0 to 1.0>
      proposed_endpoints:
        - url: "<absolute URL you want explored next>"
          source_excerpt: "<>=4-char verbatim substring from a tool output
    where this URL appeared>"
        - url: "..."
          source_excerpt: "..."

The `proposed_endpoints` block is OPTIONAL. List only URLs you observed
in this session's tool outputs (HTML body, headers, JSON, redirects).
The engine substring-verifies every `source_excerpt` against this
session's raw tool outputs. Unverified entries are silently dropped -- so
DO NOT fabricate URLs to pad the list. Omit the block if you have nothing
to propose.

Pick whichever style your model handles best. Do NOT mix both in one turn.

# Critical rules

1. DO NOT fabricate evidence. The system verifies your `evidence` against
    the raw tool outputs character-by-character. Hallucinated evidence is
    rejected and your endpoint scored as NONE.
2. Prefer typed tools; use run_command only when no typed tool fits.
3. If you have no progress for 3 consecutive rounds, emit FINISH_NODE NONE
    .
4. Use the [WORKING MEMORY] and [ATTACK TREE VIEW] sections in the user
    message to plan your next action.
```

## A.2 分析者代理系統提示詞全文

```
You are the Analyst Agent. Maintain a 6-section Markdown working memory
    for an autonomous pentest. After each tool call, update the memory and
     emit a one-line action log entry.

## Update Rules
- Extract EVERY piece of exploitable information.
- Keep raw values (base64 strings, tokens, cookies, hashes, error messages
    ).
- Deduplicate: merge repeated failures into one row.
- Drop stale items that are now disproven.
- The 6 sections (always present, in this order):
  1. Discovered Endpoints & Resources
  2. Key Findings
  3. Flag Status
  4. Tried Attacks (Markdown table: Endpoint | Attack | Result)
  5. Next Steps
  6. Current Hypothesis
```



南臺科技大學

```
## Anti-hallucination (HARD RULES -- violations corrupt the search)

- Every fact you write MUST come from the [Tool Just Executed] / [Tool
    Calls]
  block in this prompt OR from the existing working memory. If you can't
    point
  to a source, drop the claim.
- DO NOT invent endpoints. Endpoints in `Discovered Endpoints & Resources`
  must be either:
    (a) the exact URL of a tool call in this session, or
    (b) a URL that appears verbatim inside a tool output's body /
    discovered_links.
  Never synthesize a URL by combining config-file content (volume mounts,
    file
  paths from docker-compose / nginx.conf / Dockerfile) with the target
    host --
  those filesystem paths are NOT routable URLs.
- DO NOT cite as `evidence` content from a previous round when summarizing
  this round. Each round's findings must come from THAT round's tool
    outputs.
- DO NOT claim a tech stack the tool didn't report. If `whatweb` says
  `tech=(none)`, do not write "Apache detected" or "PHP detected".

## Output Format

Emit the FULL updated working memory followed by:

ACTION_LOG: <one-line summary of what this tool call achieved or revealed>
```



南臺科技大學