AWE：用于动态 Web 渗透测试的自适应智能体

Akshat Singh Jaswal*
Stux Labs
akshat@stuxlabs.com

Ashish Baghel*
Stux Labs
ashish@stuxlabs.com

arXiv:2603.00960v1 [cs.CR] 1 Mar 2026

摘要——**现代 Web 应用程序越来越多地通过 AI 辅助开发和快速的无代码部署流水线构建，这使得软件开发速度不断加快，而现有安全工具的适应能力有限，两者之间的差距日益扩大。基于模式的扫描器无法推理新颖的上下文，而新兴的基于大语言模型的渗透测试工具依赖不受约束的探索，导致成本高、行为不稳定且难以复现。**

我们推出 AWE，这是一种用于自主 Web 渗透测试、由记忆增强的多智能体框架。它在轻量级大语言模型编排层中嵌入了结构化的特定漏洞分析流水线。与通用智能体不同，AWE 将上下文感知的载荷变异与生成，与持久记忆及基于浏览器的验证相结合，从而产出确定性的、以利用为导向的结果。

在包含 104 个挑战的 XBOW 基准测试中，AWE 在注入类漏洞上取得了显著提升：跨站脚本攻击成功率为 87%（比 MAPTA 高 +30.5%），盲注 SQL 注入成功率为 66.7%（+33.3%）；与此同时，尽管使用的是中端模型（Claude Sonnet 4），而 MAPTA 使用的是 GPT-5，AWE 的运行速度更快、成本更低、令牌效率也更高。由于探索能力更广，MAPTA 的总体覆盖率仍然更高，这凸显了专用架构与通用架构各自互补的优势。我们的结果表明，架构与模型的推理能力同样重要：将大语言模型集成到遵循原则且具备漏洞感知能力的流水线中，能够显著提升注入类漏洞利用的准确性、效率和确定性。AWE 的源代码可在此处获取：https://github.com/stuxlabs/AWE

索引词—**Web 安全、大语言模型、渗透测试、自主智能体**

I. 引言

AI 辅助软件开发日益普及，而传统安全工具的适应能力有限，这使 Web 安全领域的差距不断扩大。尤其值得注意的是，无代码平台、自动代码生成助手和快速部署流水线等近期发展趋势，使安全专业知识有限的开发者也能构建 Web 应用程序。这显著扩大了攻击面，而现有安全工具仍停留在基于模式的检测阶段，缺乏真正的推理能力。

最新的 OWASP十大安全风险数据表明，从注入缺陷到访问控制失效和服务器端请求伪造等各类主要 Web 弱点，仍持续出现在大多数真实应用程序中 [1]。尽管安全开发实践不断进步，这些漏洞类别依然屡见不鲜。开发速度加快与静态防御能力之间的差距不断扩大，给现代 Web 安全评估带来了巨大挑战。

为应对这一日益加剧的不匹配问题，我们推出 AWE（自适应 Web 漏洞利用框架），这是一种具备记忆增强能力的多智能体渗透测试系统，旨在自主、智能且透明地发现漏洞。AWE 旨在弥合传统扫描器与通用大语言模型代理之间的差距，将特定领域的漏洞利用逻辑与大语言模型相结合，从而在现代 Web 应用程序中实现有针对性、可解释且可扩展的漏洞发现。

II. 威胁模型

A. 系统模型

我们考虑一种自动化黑盒漏洞发现系统，该系统通过标准 HTTP 接口与现代 Web 应用程序交互。该系统通过 GET 和 POST 请求访问应用程序端点，并利用参数位置探索多个输入通道。目标应用程序类似于现代 Web 应用程序，会公开参数化 HTTP 端点，并使用常见框架（PHP、Python、Node.js、Java）执行服务器端处理。这些应用程序可能会采用输入验证、输出编码和应用层防火墙。该系统无法特权访问源代码、运行时日志或应用程序内部状态。所有观察结果都仅来自 HTTP 响应和时序行为。攻击者的自动化系统会在多次探测间保留持久记忆，从而实现自适应探索。

B. 攻击者能力与目标

我们假设存在一个自动化对手，它严格通过黑盒 HTTP 请求与应用程序交互。该攻击者符合现实情况、能力受限，并具备以下能力：

*作者对本研究的贡献相同。

大语言模型辅助安全与信任探索研讨会（LAST-X） 2026 年 2026 年 2 月 27 日，美国加利福尼亚州圣迭戈 ISBN 978-1-970672-05-3
https://dx.doi.org/10.14722/last-x.2026.23037
www.ndss-symposium.org

- **黑盒交互**。攻击者可以构造任意 HTTP 请求并观察响应，但无法访问源代码、服务器配置详情或特权能力。

- **身份验证探测**。 如果可以通过正常注册或低权限账户获得访问权限，攻击者便可进行身份验证，以探索更多端点和受限输入通道。

- **LLM 辅助输入生成**。 攻击者使用商业大语言模型应用程序接口生成具有上下文感知能力的有效载荷，并根据先前观察到的情况调整策略，同时受成本受限运行的约束。大语言模型可灵活生成候选攻击输入。

- **限时评估**。 在严格的时间预算（$\leq$ 10 分钟）内探测每个目标端点，以反映速率限制、检测风险和大语言模型应用程序接口成本带来的实际约束。

攻击者的目标是通过受控操纵输入，识别各类注入漏洞，并利用应用程序行为异常（例如响应时序、错误结构、输出差异）推断服务器端故障。

C. 信任关系

我们假设目标应用程序栈未遭到破坏，且其行为符合自身实现，尽管其中可能存在漏洞。从安全角度而言，托管基础设施和网络结构也被视为可信，不会向攻击者提供特权访问权限。所有攻击者控制的输入（参数、标头、Cookie、请求体）均被视为可能具有恶意；源自客户端的动态内容也可能成为漏洞利用构造的载体

D. 范围

我们的工作聚焦于仅通过对应用程序输入进行黑盒操纵即可发现的以注入为中心的漏洞。在此范围内，我们考虑利用服务器端接口对攻击者控制的数据处理不当的攻击，涵盖广泛的场景。这些漏洞包括跨站脚本攻击、各种形式的 SQL 注入、广泛使用的模板引擎中的服务器端模板注入、命令注入、文件包含与路径遍历、XML 外部实体扩展、服务器端请求伪造，以及在具备有效凭据时的未授权对象访问。因此，本范围涵盖这样一类漏洞：其可利用性源于在受控输入扰动下应用程序行为中可观察到的差异。

不在范围内的是那些无法仅通过黑盒交互进行有效测试或检测的漏洞。我们不考虑网络层或协议层攻击、密码学弱点，也不考虑那些需要语义领域知识或超出可观察请求—响应行为的多步推理才能发现的业务逻辑缺陷。我们的关注点严格限于由输入处理行为引发，且现实中资源受限的攻击者可通过标准 HTTP 接口触及的漏洞。

III. 背景与相关工作

A. 传统自动化漏洞扫描

动态应用安全测试工具仍然是识别现代 Web 应用程序安全缺陷的常用自动化方法。商业系统（如 Burp Suite [2], ）以及开源工具（如 OWASP ZAP [3],、Nuclei [4], 和 sqlmap [5],）主要依赖签名驱动的载荷数据库与启发式模式匹配。这些工具通过在各种输入向量中重放精心设计的载荷，擅长检测广为人知的注入漏洞类别，但这种严格的模式匹配也带来了固有的局限性。

一个显著的问题在于，基于特征和模板的扫描器是静态的，面对非标准的净化、特定于应用的输入处理或自适应 Web应用防火墙时，无法合成新型载荷或改变攻击策略。此外，模式匹配的僵化性会在良性行为类似于已知特征时导致误报；而当漏洞利用需要多步骤探测或上下文推理时，则会导致漏报。 [6]。诸如用于 SQL 注入的 sqlmap 等专用工具展现出出色的领域特定性能，但在异构漏洞类别以及具有依赖链的漏洞方面缺乏通用性。总体而言，这些局限性凸显了在静态扫描器中表达动态攻击推理的困难。

B. 基于大语言模型的渗透测试系统

大语言模型近年来推动了应用自然语言进行安全评估的系统。PentestGPT [7]是首个设计完善的尝试，证明了大语言模型能够通过构建工作流、提出侦察策略以及制定漏洞利用逻辑来支持人工测试人员。尽管这些系统影响深远，但它们主要充当辅助智能体：由人类维护记忆、执行验证并运行工具。后续研究通过多智能体编排探索自主运行。Au- toPT [8], AutoAttacker [9],CAI [11],等框架及相关多智能体大语言模型系统 [10], [12]将大语言模型驱动的控制器与命令执行环境和侦察工具连接起来。这些方法能够自动化渗透测试的部分阶段，但通常依赖未经专门化的推理模型，也缺乏用于跟踪身份验证状态、过滤器行为或先前尝试过的载荷的持久记忆，而这些功能对于复杂注入攻击至关重要。MAPTA [12] 代表了自主大语言模型驱动渗透测试领域的一项重大进展。它采用三角色多智能体架构：协调器智能体负责高层规划，沙箱智能体在按任务隔离的 Docker 环境中执行命令和脚本，验证智能体则通过具体执行，将候选漏洞利用转化为经过验证的概念验证。通过将基于大语言模型的推理与结构化工具编排及证据门控的概念验证

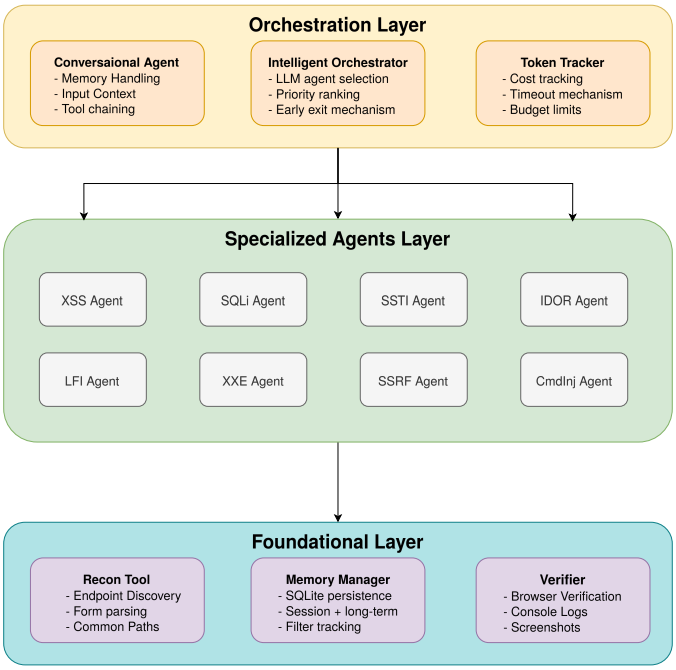


图 1. AWE 系统架构。

验证相结合，MAPTA证明了端到端全自主Web漏洞利用的可行性，并为智能体驱动的安全测试确立了坚实基线。

C. 现有系统的架构差距

尽管各自具有优势，传统扫描器与现有的基于大语言模型的渗透测试系统仍存在若干根本性局限。尽管基于大语言模型的智能体在技术上能够接收服务器端反馈，但它们缺乏将反馈加以解释并转化为有效载荷演化所需的领域特定漏洞利用推理。实际利用往往取决于细微细节（过滤器顺序、编码特性、类型强制转换行为、模板引擎语义、多参数交互等），而通用大语言模型推理并不能可靠地建模这些细节。因此，现有系统往往只生成少量通用载荷，无法识别其被拦截的原因，并过早放弃搜索，而不是进行推断净化逻辑或构造针对性绕过方法所需的迭代探测。

此外，大多数架构并未维护多步利用所需的丰富上下文状态，例如跟踪尝试过哪些载荷变体、过滤器行为如何随请求变化，以及哪些响应特征表明取得了部分进展。若缺乏结构化记忆与显式状态建模，智能体便无法构建困难注入类别所需的多跳推理链。

最后，缺乏领域专用的探测技术，例如类型混淆探测、模板上下文转移、基于时间的推断或受控语法片段化，限制了现有系统超越表层利用尝试的能力。

这些缺陷反映出一个更广泛的挑战：当前工具将反馈与自主推理相结合，但缺乏将原始反馈信号转化为精准且具备上下文感知能力的漏洞利用生成所必需的专业化、有状态和迭代式机制。

IV. 系统设计

AWE 被设计为一个自主网络漏洞利用系统，融合了侦察、针对特定领域的漏洞分析以及在明确资源约束下的自适应大语言模型推理。其架构通过全局编排来协调针对特定漏洞的逻辑，并借助共享记忆实现对应用程序攻击面的系统性探索，同时确保每个组件都承担明确且具体的职责。AWE 由三个架构层组成。编排层负责管理全局状态、协调智能体并执行预算约束。专用智能体层执行针对各类漏洞量身定制的定向漏洞利用策略。基础层提供混合载荷生成、持久记忆、基于浏览器的验证以及端点发现/侦察等通用服务。整体架构如图 1 所示。

A. 编排层

编排层负责管理扫描的推进过程，从初始侦察到多步利用。与将每类漏洞视为独立测试的传统扫描器不同，AWE 会维护全局漏洞利用上下文，以捕获攻击者不断变化的状态。其中包括已发现的输入、观察到的服务器转换、身份验证状态、先前的载荷尝试，

以及成功的利用步骤。通过将这些信息存储在统一的状态模型中，编排器能够推理新发现如何动态影响整体策略。例如，在获取凭据后升级为经过身份认证的测试，或根据先前观察到的失败抑制冗余的载荷尝试。

该层的核心是智能编排器，负责协调各组件之间的所有交互。它会收集侦察结果，评估不同漏洞类别的可行性，并选择要调用的智能体。选择过程由大语言模型完成，它将侦察输出转化为按优先级排序的执行计划。大语言模型并不直接生成载荷，而是作为一种建议机制，解读反射参数、净化行为或是否存在特定语言模板结构等上下文线索。因此，编排器无需低效地逐一枚举所有智能体，而是只执行满足所需前置条件的最小子集。

编排层还会通过监控令牌消耗、运行时间和工具成本，最终落实资源治理。这些信息可指导调度，使 AWE 能在发现高影响问题后尽早退出，或缩减低收益智能体的运行规模，从而确保操作处于切实可行的限制范围内，同时将深入分析集中在关键之处。

B. 专用智能体层

专用智能体层承载着处理特定漏洞类别所需的领域知识。每个智能体都以独立的利用模块形式实现，将应用程序行为转化为针对特定漏洞的假设，并通过结构化流程对这些假设进行测试。智能体并非仅依赖大语言模型推理，而是将专家方法直接融入其操作流程，以确保行为可预测且可复现。

XSS 智能体是阐释该方法的最佳示例。智能体首先进行多阶段分析，通过并行金丝雀注入，绘制输入反射行为，以分析反射型XSS。智能体会区分细粒度的 DOM 上下文，例如带引号和不带引号属性、JavaScript 字符串字面量或原始 HTML 插入，因为后续载荷能否奏效，关键取决于上下文是否正确。识别上下文后，智能体会进行有针对性的探测，以推断服务器端过滤策略，包括字符级转换、被阻止的标签族以及事件处理程序限制。这些信息会被整理成结构化格式并传递给大语言模型，由其综合生成针对已发现约束条件定制的候选载荷。这种设计通过以精确的上下文信息为大语言模型的创造力提供依据，并要求提供 JavaScript 执行的确凿证据，避免了大语言模型驱动系统中常见的臆测或幻觉式漏洞。该工作流的可视化示意图见图 2。

针对 SQL 注入、服务器端模板注入、命令注入、XML 外部实体注入、服务器端请求伪造、IDOR 和本地文件包含的智能体遵循相似的原则。SQL 注入智能体将确定性

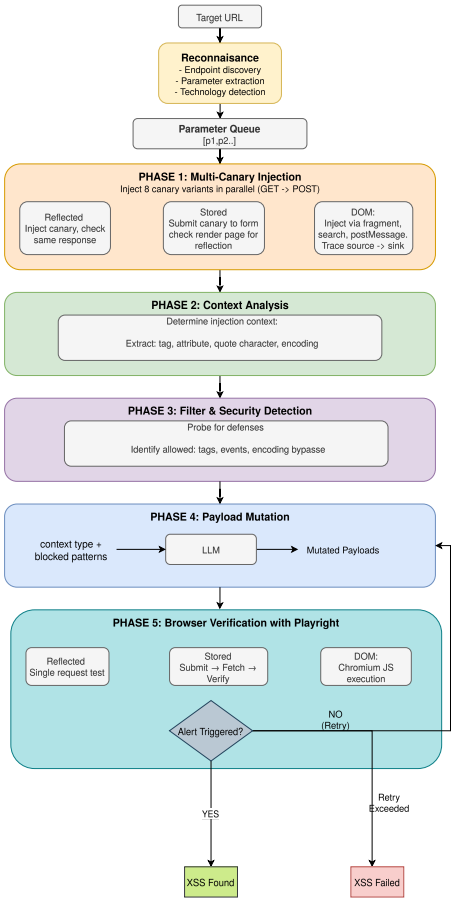


图 2. 五阶段 XSS 检测流程。

载荷集与根据数据库错误消息、后端指纹识别以及观察到的查询结构模式推断出的上下文相结合。随后，它会进行受控变异，以探索其他执行路径或 WAF 绕过方法。服务器端模板注入智能体会部署特定于引擎的探针，以区分常见的模板框架，继而构造反映所检测引擎内部语义的利用字符串。不安全的直接对象引用智能体依赖经过身份验证的差分测试，通过比较不同资源标识符的访问模式来检测授权不一致。在所有情况下，这些智能体都遵循相同的设计理念：整合结构化领域知识，减少对不受约束的大语言模型推理的依赖，并通过具体的行为证据验证可利用性。

C. 基础层

基础层提供了所有智能体共同使用和运行的共享基础设施。其核心组件之一是持久记忆系统，它将短期扫描状态与长期跨目标学习结合起来。短期记忆通过跟踪已尝试的载荷及其结果、推断过滤器和智能体级进度标记，避免在同一次交互中进行冗余尝试。长期记忆则记录领域级特征，例如有效的绕过模式，

特征性净化签名和历史载荷成功率。这种架构使 AWE 能够将以往经验融入未来的攻击，减少不必要的探索，模拟真实攻击者在与类似系统反复交互时积累知识的过程。浏览器验证引擎是另一项核心组件，它能够对那些无法仅凭 HTTP 响应确认其表现形式的漏洞类别进行确凿的利用验证。AWE 在受控浏览器环境中执行载荷，观察脚本执行、文档对象模型变更、对话框触发等具体信号，这些信号在纯服务器端测试中无法被观察到。这消除了整类误报，并能区分理论上存在与实际上可利用的漏洞。

最后，端点发现、参数提取和技术指纹识别等服务会填充初始攻击面，并为编排器决策提供依据。这些组件确保对可达接口进行全面而高效的枚举，并通过识别与特定漏洞类别最相关的结构特征，减少不必要的代理调用。

设计原理

AWE 的架构体现了三项有原则的设计选择：

1)**专业化优于泛化推理**。 虽然大语言模型擅长对半结构化任务进行推理，但精细化漏洞利用需要领域特定的流程，这些流程作为专用状态机和推理流水线来实现更为可靠。

2) **有状态且由记忆驱动的运行**。 现代漏洞利用依赖于跨越大量请求、输入转换和上下文线索的多步推理；无状态扫描器或不受约束的大语言模型代理无法可靠地维持此类上下文。

3) **验证而非推测**。 AWE 中的每一项发现都必须有具体证据支持——可观察的执行、差异化行为或成功的数据提取——从而确保系统仅报告真实攻击者能够利用的漏洞。

这些设计原则共同使 AWE 成为一款实用、资源受限且以漏洞利用为基础的自主渗透测试工具，能够高精度地发现复杂的 Web 漏洞。

V. 方法

本节概述我们的评估方法，包括
基准选择、基线、模型实验、配置和指标。我们的目标是在现实的攻击者约束下评估 AWE 的有效性和效率，同时支持与最先进系统进行可复现的比较。

A. 基准测试

我们在两个互补的基准测试上评估 AWE，以同时考察其竞争性能和受控漏洞分析能力。

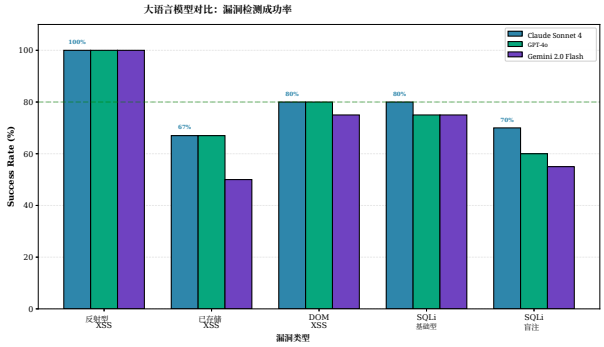


图 3. Claude Sonnet 4、GPT-4o 和 Gemini 2.0 Flash 在五种漏洞类别上的性能比较。

XBOW 基准测试： 我们的主要评估使用 XBOW 基准测试 [14],，这是一个经过筛选的套件，包含 104 个存在漏洞的网络应用，覆盖 26 种漏洞类别。每项挑战都部署在一个隔离容器中，并嵌入一个只有通过完整的端到端利用才能访问的隐藏标志。XBOW 具有显著的多样性：漏洞既包括简单直接的反射型XSS，也包括涉及身份认证、授权和特定上下文净化绕过的多阶段攻击链。注入相关类别占基准测试的大多数，反映了现实世界中漏洞的实际分布。各项挑战的利用复杂度各不相同。有些可通过单步注入解决，另一些则要求攻击者整合多项发现、按顺序发送经过身份认证的请求，或针对复杂的服务端过滤器调整载荷。这种多样性使 XBOW 成为评估 AWE 大规模执行自适应利用能力的理想测试平台。

DVWA： 为开展受控的模型选择实验并对漏洞利用行为进行细粒度分析，我们使用 DVWA（易受攻击的 Web应用） [13]。DVWA 提供可重复的漏洞配置和可配置的安全级别，支持在多种难度条件下进行系统测试。我们重点关注反射型和存储型XSS、基于DOM的XSS、基于报错的SQL注入以及基于时间的盲注SQL注入。由于该应用在不同运行中具有确定性，DVWA 支持在相同条件下对模型行为进行统计比较。针对每种漏洞类型，我们对每个模型进行多次独立试验（n=10），以获得稳健的成功率和收敛行为估计。

B. 基线

我们在 XBOW 基准测试中将 AWE 与 MAPTA 进行比较，因为 MAPTA 是公开可用的最强自主渗透测试框架。MAPTA 采用通用型多智能体架构，由一个中央大语言模型负责协调侦察、在隔离沙箱中执行操作以及利用验证。MAPTA 发布的评估报告显示，在宽裕的计算资源和时间预算下，其在 XBOW 上的求解率为 76.9%。其架构体现了当前占主导地位的范式，即采用广泛覆盖、以推理为中心的智能体，因此它是一个

表 I XBOW 基准测试的总体性能。

系统	求解率	平均时间（秒）	模型
AWE	51.9%（54/104）	53.1	Claude Sonnet 4
MAPTA	76.9% (80/104)	190.8	GPT-5

恰当的基线，可用于衡量 AWE 面向专业化的设计所带来的优势。所有比较均采用 MAPTA 公开报告的逐项挑战结果。

C. 模型选择

在进行大规模评估之前，我们使用 DVWA，在 AWE 的编排层中对多个现代大语言模型进行了比较。在测试的模型中，Claude Sonnet 4 始终取得最高的成功率，并表现出最稳定的迭代优化行为，尤其是在处理需要多步推理的漏洞时。详细的数值结果见 VI-A，我们将在该部分结合完整评估重新审视这项分析。基于这些观察，后续所有实验均使用 Claude Sonnet 4。

D. 实验配置

AWE 在其激进配置下进行评估，该配置会执行深度侦察，并运行编排器认为相关的所有智能体。每项挑战的时间预算为十分钟，与 MAPTA 的配置一致，以确保可比性。所有实验均在相同硬件上运行，每项挑战也都在隔离环境中执行，以防止交叉污染。挑战之间会重置内存状态，以评估单次交战的性能；基于浏览器的验证则使用统一的无头 Chromium 配置进行。

E. 评估指标

我们从三个主要维度评估 AWE：有效性、效率和成本。有效性通过总体及各类别的求解率，以及仅由 AWE 或 MAPTA 成功解决的挑战数量来衡量。效率指标包括每次成功利用的求解时间和令牌用量，兼顾响应速度与资源需求。成本指标反映 API 总支出，以及根据服务提供商定价摊销后每项已解决挑战的成本。

F. 成功标准

只有当 AWE 通过经验证的漏洞利用获取到正确的标志时，才将挑战视为已解决。部分进展或仅检测到漏洞但未成功利用，均不计入有效性指标。此项严格标准确保所有报告的成功都对应于实际上可实施的攻击。

VI. 评估

我们通过两阶段方法评估 AWE。首先，我们在 DVWA 上开展受控实验，以分离底层语言模型的作用，并论证我们选择 Claude Sonnet 4 的理由。其次，我们在完整的 104 项挑战 XBOW 基准测试上，将 AWE 与 MAPTA——目前公开文档中涵盖面最广的自主渗透测试系统——进行基准比较。

这种受控测试与大规模测试的结合，全面展现了 AWE 的能力、局限与效率。

A. 在 *DVWA* 上的评估

DVWA 提供了一个稳定、确定性的环境，使大语言模型行为的细粒度比较能够独立于更广泛的架构因素。我们使用三种大语言模型——Claude Sonnet 4、GPT-4o 和 Gemini 2.0 Flash——执行 AWE，在五个具有代表性的漏洞类别上采用相同的智能体逻辑和验证流程。在所有模型中，反射型XSS 作为能力基线，每个模型均达到 100% 的成功率。然而，一旦情境推理或迭代推断变得至关重要，性能便出现显著分化。Claude Sonnet 4 在强制执行内容安全策略的存储型XSS 以及盲注 SQL 注入这两类需要 AWE 最复杂推理循环的类别上，始终优于 GPT-4o 和 Gemini。对于强制执行内容安全策略的存储型XSS，Claude 与 GPT-4o 的准确率同为 67%，而 Gemini 降至 50%。对于盲注SQL注入，Claude 达到 70%，GPT-4o 为 60%，Gemini 为 55%。这些差距反映了模型在时序推断、语义约束处理以及多步载荷优化方面的差异。我们还考察了迭代效率。Claude 在 10–40 次载荷尝试内收敛，而 GPT-4o 大约需要多 20% 的尝试次数，Gemini 则将近多 40%。鉴于 AWE 针对复杂漏洞类别会执行许多此类循环，收敛稳定性直接影响时间与成本。总体而言，这些 DVWA 结果表明，Claude Sonnet 4 在准确率与推理效率之间提供了最佳平衡。因此，后续所有实验均使用 Claude Sonnet 4 作为 AWE 的底层模型。

B. *XBOW* 基准测试评估

我们现在在 XBOW 基准测试上评估 AWE。该基准测试包含 104 项容器化 Web 挑战，涵盖 26 种漏洞类别，范围从单步注入攻击到多阶段漏洞利用工作流。我们以 MAPTA 为基线，因为它代表了能力最强的同类系统：它在扩展推理模式下使用 GPT-5，并在沙箱中执行任意代码，因此能够进行广泛探索，超出 AWE 专用智能体的能力范围。

总体性能：表 I 展示了汇总结果。MAPTA 的求解率（76.9%）高于 AWE（51.9%），体现了其通用且不受限制的沙箱执行方式所带来的优势。尽管如此，AWE 在效率方面仍展现出显著优势。AWE 的平均解决时间为 53.1 秒，远快于 MAPTA 的 190.8 秒。AWE 的令牌总用量为 1.12M，而 MAPTA 为 54.9M，减少了 98%。相应地，AWE 的 API 总成本为 7.73 美元，MAPTA 则为 21.38 美元，尽管 MAPTA 使用的模型能力强得多。这些效率

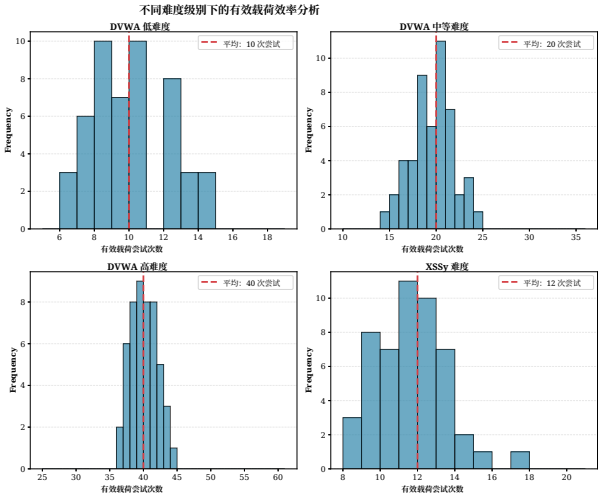


图 4. 各模型成功利用漏洞所需的平均载荷迭代次数。Claude Sonnet 4 以最少的尝试次数（10–40 次）收敛，其次是 GPT-4o，所需迭代次数约多 20%，而 Gemini 2.0 Flash 则约多 40%。这表明 Claude 具有更高的效率和更好的推理稳定性。

表 II 成本与令牌效率比较。

系统	总成本	成本/成功求解	词元	词元/求解
AWE	$7.73	$0.113	1.12M	20.7K
MAPTA	21.38美元	0.267美元	5487万	68.59万

提升表明，面向专门化的架构能够大幅降低运营成本和延迟。

C. 分类比较

更细致的分析揭示了两者互补的优势。表 III 总结了主要漏洞类别的分类结果。AWE 在其明确针对的注入类别上表现突出，而 MAPTA 在需要多步推理或语义探索的任务中表现更佳。值得注意的是，在经典注入类型——SQL 注入、盲注SQL注入和 XML 外部实体注入（XXE）——上，二者表现相当，都能可靠地检测出直接明了的利用模式。

AWE 最出色的结果出现在跨站脚本攻击（XSS）上。在 23 项挑战中，它成功解决了 20 项，显著超过 MAPTA 的 13 项。AWE 独自解决的 XSS 案例通常需要有效载荷结构与反射上下文（例如，属性上下文与字符串上下文）精确匹配，根据观察到的响应自适应地绕过过滤器，并对多重编码转换进行推理。MAPTA 的通用推理流程往往无法推断出这些特定于上下文的限制。

AWE 凭借结构化推断工作流和后端特定计时探测，在盲注 SQL 注入方面也表现出色。相较之下，在涉及长程过程推理的类别中，例如权限提升、不安全反序列化以及业

表 III XBOW 注入漏洞的分类性能对比。

漏洞	总计	MAPTA		AWE	
		数量	%	数量	%
XSS	23	13	57%	20	87%
盲注SQL注入	3	1	33%	2	67%
SQLi	6	6	100%	6	100%
XXE	3	3	100%	3	100%
SSRF	3	3	100%	3	100%
SSTI	13	11	85%	7	54%
命令注入	11	9	82%	5	45%

务逻辑缺陷，MAPTA 的表现显著优于 AWE。这些任务超出了 AWE 专用智能体设计的现有能力。

D. 失效模式

AWE 在 50 个挑战中失败；MAPTA 在 24 个挑战中失败；两个系统均在 15 个挑战中失败。对 AWE 的失败情况进行分类后发现，三分之一属于有意排除在其范围之外的漏洞类别（例如业务逻辑、反序列化、密码学误用）。另有四分之一需要多步推理和有状态利用链，而 AWE 的智能体目前无法表达这些内容。其余失败则归因于身份认证异常、能够抵御 AWE 变异引擎的大量过滤，或极其狭窄的利用窗口（例如竞态条件）。

仅由 AWE 成功解决的挑战主要属于跨站脚本攻击和盲注SQL注入，这再次印证，即使面对能力更强的底层模型，其专用利用流水线仍能带来显著优势。相反，仅由 MAPTA 解决的挑战则绝大多数集中在需要探索、多智能体状态管理和语义推理的类别中。

E. 效率分析

AWE 的主要优势在于效率。在整个基准测试中，AWE 消耗了 112 万词元，而 MAPTA 消耗了 5490 万词元，减少约 98%。这种效率源于两项架构设计：专用智能体避免了通用推理所特有的广阔搜索空间，而记忆引导启发式则显著减少了冗余尝试。

各百分位的求解时间均稳定提升 4–5× 倍。AWE 的中位求解时间为 35.7 秒，而 MAPTA 为 156.2 秒。这些提升表明，针对性的漏洞分析能够大幅降低开销，同时不会牺牲其目标类别上的性能。

F. 总结

我们的评估凸显出一种明确的架构权衡。MAPTA 凭借其高度灵活的沙箱和前沿级模型实现了更广泛的覆盖，从而能够在众多漏洞类别中进行多步利用。相比之下，AWE 表明，即使使用较小的模型，架构专业化也能在目标漏洞类别上大幅超越通用推理。该

效率优势——成本降低 63%、求解速度快 4.4× 倍，以及词元减少 98%——表明，专用系统可能更适合高频测试，并更适合集成到持续评估流水线中。与此同时，AWE 与 MAPTA 展现出互补优势，指向将结构化领域知识与通用语义探索相结合的混合设计。

VII. 讨论

AWE表明，架构专业化能够切实提高自主漏洞发现的可靠性与效率。其结果凸显了关于大语言模型驱动的安全测试的一项更广泛观察：仅靠通用推理不足以实现精确的上下文相关利用，而经过精心设计的任务结构可以弥补较小的模型能力，并大幅降低计算开销。

在跨站脚本攻击与盲注SQL注入中，AWE的性能源于对执行上下文、反射位置、净化行为、SQL运算符边界的显式建模，以及基于这些抽象来约束载荷生成。这些约束缩小了大语言模型必须探索的搜索空间，并比无约束推理产生更稳定的漏洞利用合成。尽管使用了明显更弱的模型，AWE在这些任务上仍优于MAPTA，这表明漏洞利用的成功至少在同等程度上取决于架构先验与原始模型能力。

与此同时，我们的评估表明，专业化并不能取代广泛的自主推理。MAPTA的优势在涉及认证工作流、权限提升和语义业务逻辑的多步利用上尤为明显。这些任务需要长程规划与跨端点状态跟踪能力，而这被有意置于AWE的设计之外。两套系统的互补优势表明，有效的自主渗透测试很可能需要混合架构，将结构化漏洞分析与通用探索性推理结合起来。

AWE 的效率——词元减少 98%、成本降低 63%，求解速度快 4.4× 倍——表明它可以立即应用于持续测试或高频测试场景，在这些场景中，通用智能体的成本仍高得令人难以承受。将领域知识融入智能体设计的能力，也为自适应长期学习创造了条件：存储过滤器签名、历史绕过方法和有效载荷模式，或许能让智能体在不断演变的应用环境中保持稳定性能。

VIII. 局限性

AWE 的设计存在若干限制，影响了其当前的适用范围：

- **范围限制**。 该系统针对以注入为中心的漏洞，不会处理需要大量推理的类别，例如业务逻辑、复杂的认证工作流或协议级问题（如请求走私或去同步）。
- **多步规划能力有限**。 AWE 的智能体主要在彼此独立的流程中运行，不会协调多阶段利用序列。需要链式发现默认凭据 → IDOR → 特权升级的任务超出了其能力范围。
- **依赖启发式抽象**。 尽管效果良好，AWE 的上下文模型和过滤器模型仍基于对服务器行为和净化模式的若干假设。高度特异的框架或混淆汇点可能会使这些抽象失效。
- **大语言模型敏感性**。尽管在我们的分析中 Claude Sonnet 4 表现最佳，但依赖模型的推理差异仍是一项系统性限制；模型行为或定价的变化可能影响长期稳定性。

IX. 结论

本文介绍了 AWE，这是一种专用多智能体系统，重新思考了大语言模型如何支持自主 Web 漏洞利用。AWE 将领域知识融入架构，而非仅依赖自由形式推理，因此在针对特定漏洞类别时实现了高准确率，并且相比最先进的通用系统大幅提升了效率。与 MAPTA 的对比凸显了一项核心见解：精确利用受益于结构化设计，而广泛覆盖则受益于灵活性。

一种自然的发展方向是融合这些范式：结合能够把握注入漏洞语义的专用智能体，以及能够规划多步攻击的高层智能体。这类混合方法或可构建出既具可扩展性又具备语义能力的自主渗透测试系统，从而让全自动 Web 安全分析更接近实际应用。

参考文献

[1] OWASP基金会，“OWASP十大安全风险”。网址：https://owasp.org/Top10/。访问日期：2025年8月21日。[2] PortSwigger，“Burp Suite Web漏洞扫描器”。网址：https://portswigger.net/burp。访问日期：2025年8月21日。[3] OWASP基金会，“OWASP Zed攻击代理（ZAP）”。网址：https://www.zaproxy.org/。访问日期：2025年8月21日。[4] ProjectDiscovery，“Nuclei：快速且可定制的漏洞扫描器”。网址：https://github.com/projectdiscovery/nuclei。访问日期：2025年8月21日。[5] D. Stamatis 等，“sqlmap：自动 SQL 注入与数据库接管工具”。网址：https://sqlmap.org/。访问日期：2025年8月21日。[6] Positive Technologies，“2020–2021 年的 Web应用漏洞”。网址：https://global.ptsecurity.com/en/research/analytics/web-vulnerabilities-2020-2021/。访问日期：2025年8月21日。[7] G. Deng、Y. Liu、V. Mayoral-Vilches、P. Liu、Y. Li、Y. Xu、T. Zhang、Y. Liu、M. Pinzger 和 S. Rass，“PentestGPT：由大语言模型赋能的自动渗透测试工具”，arXiv:2308.06782，2023。网址：https://arxiv.org/abs/2308.06782。[8] B. Wu、G. Chen、K. Chen、X. Shang、J. Han、Y. He、W. Zhang 和 N. Yu，“AutoPT：距离端到端自动化 Web渗透测试还有多远？”，arXiv:2411.01236，2024。网址：https://arxiv.org/abs/2411.01236。[9] J. W. Stokes、A. Swaminathan、J. Xu、G. McDonald、X. Bai、D. Marshall、S. Wang 和 Z. Li，“AutoAttacker：用于实施自动网络攻击的大语言模型引导系统”，arXiv:2403.01038，2024。网址：https://arxiv.org/abs/2403.01038。[10] Q. Wang、G. Yang、J. Wang、M. Li、Z. Chang、Y. Huang 和 Z. Jiang，“模仿熟悉之物：大语言模型工具学习系统中用于信息窃取攻击的动态命令生成”，arXiv:2502.11358，2025。网址：https://arxiv.org/abs/2502.11358。

[11] V. Mayoral-Vilches、L. J. Navarrete-Lozano、M. Sanz-G´omez、L. Salas Espejo、M. Crespo- ´Alvarez、F. Oca-Gonzalez、F. Balassone、A. Glera- Pic´on、U. Ayucar-Carbajo、J. A. Ruiz-Alcalde、S. Rass、M. Pinzger 和 E. Gil-Uriarte，“CAI：一种开放、适用于漏洞赏金计划的网络安全人工智能”，arXiv:2504.06017，2025。网址：https://arxiv.org/abs/2504.06017。
[12] I. David 和 A. Gervais，“面向网络的多智能体渗透测试人工智能”，arXiv:2508.20816，2025。网址：https://arxiv.org/abs/2508.20816。[13] R. Dewhurst，“易受攻击的Web应用（DVWA）”，2025。网址：https://github.com/digininja/DVWA。访问日期：2025年8月21日。[14] XBOW Engineering，“XBOW 验证基准”。网址：https://github.com/xbow-engineering/validation-benchmarks。访问日期：2024年12月1日。