XSSplore: A Tree-Structured Prompting and Actor-critic Framework for Autonomous Red Teaming XSS Payload Generation

Gang Yang
Information Support Force Engineering University
Wuhan, China
yanggang11@nudt.edu.cn

Bo Wu
Information Support Force Engineering University
Wuhan, China
cherry_wb@163.com

Tao Xia
Information Support Force Engineering University
Wuhan, China
xiatao17@nudt.edu.cn

***Abstract*—Autonomous red teaming penetration testing has been of great significance in ensuring the security of Web systems, particularly in the face of cyber threats such as XSS attacks. However, in real-world scenarios, it often involves contending with complex Web Application Firewalls (WAFs), which necessitates substantial expert efforts for automated methods in practical applications. Consequently, autonomously generating bypassable XSS payloads capable of handling adversarial environments remains a challenge. To address this issue, this paper proposes *XSSplore*, a method that combines tree-structured prompting and an actor-critic architecture, based on plug-in knowledge and a corpus. By mapping knowledge onto tree-structured prompting paths, *XSSplore* can more effectively explore and exploit black-box environments while selecting appropriate bypass strategies. Additionally, due to the predefined corpus and the actor-critic mechanism, *XSSplore* can effectively ensure the executability of generated samples and mitigate the impact of hallucinations. We conducted experiments on 20 vulnerable Web applications within environments equipped with WAFs. The experimental results demonstrate that *XSSplore* can accomplish autonomous and effective XSS payload generation tasks, significantly outperforming existing methods.**

***Index Terms*—software vulnerability, pentesting, cross-site scripting, large language model, prompting engineering, code generation.**

I. Introduction

Cross-Site Scripting (XSS) remains one of the most prevalent threats to Web applications, consistently ranking high on the OWASP list. Attackers inject malicious scripts into Web pages, leading to data breaches and system hijacking [1]. Red teaming penetration tests [2] after deployment can effectively discover XSS vulnerabilities, guiding remediation to mitigate real-world cyber risks.

In practical red teaming scenarios, conducting XSS attack emulations typically requires expert experience to analyze vulnerabilities and construct test cases. Because red teaming is predominantly black-box (testers lack source code or database access), manual efforts reduce testing efficiency. Hence, automated XSS payload generation is needed.

Gang Yang is the corresponding author.

Existing automated methods fall into three categories. Template-based methods rely on predefined rules but fail against adversarial scenarios, especially Web Application Firewalls (WAFs). Heuristic-based mutation methods [3], [4] offer flexibility yet generate limited payloads due to confined search spaces and blind concatenation. Generative methods using reinforcement learning [5], [6], neural networks, or language models [7] produce high-quality payloads but depend on high-quality training sets, making it challenging to generate bypassable payloads in practical adversarial settings.

Large Language Models (LLMs) bring powerful contextual understanding and reasoning capabilities for red teaming [8]–[10], enabling agile generation of effective XSS payloads [11], [12]. However, current LLM-based methods [7] mainly use sequential or single-turn attempts, which are prone to generating ineffective tests in adversary-dense environments, still requiring expert remediation.

To address this challenge, we propose *XSSplore*, an agentic workflow with tree-structured prompting [13], [14] to iteratively refine payloads tailored to diverse scenarios. Inspired by prior work [15], we incorporate a self-critique mechanism enabling rollback operations to mitigate hallucinations. Our main contributions are:

- A plug-in knowledge set and corpus that minimize fine-grained generation errors.
- A tree-structured prompting framework that emulates human experts' step-by-step payload construction for deep adversarial penetration testing.
- An actor-critic LLM agent that evaluates payload quality and triggers rollbacks, reducing error accumulation and hallucinations from long-chain prompting.

II. Proposed Method

A. Overview

The proposed *XSSplore* consists of three primary components: plug-in knowledge set and corpus, tree-structured prompting strategy, and actor-critic architecture for reliable XSS payload generation, as illustrated in Fig. 1.

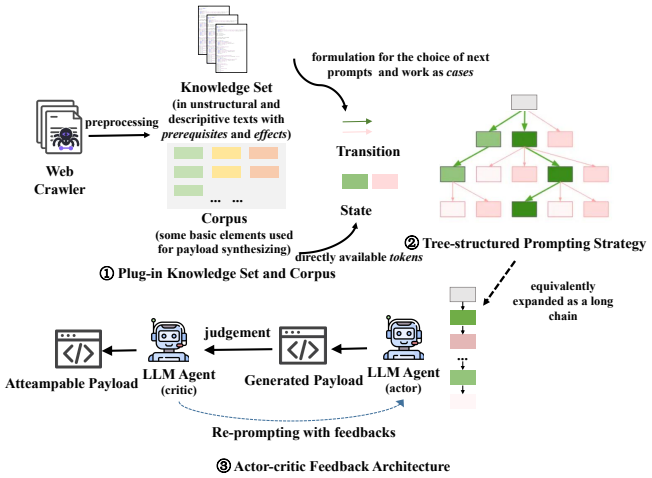

Fig. 1. An overview of proposed *XSSplore*.

The plug-in knowledge Set offers path planning-support for LLM reasoning, while corpus provides basic lexical elements. The tree-structured prompting strategy empowers the LLM to select paths based on feedback from multi-turn progressive attempts. This enables the construction of XSS payloads to employ more sophisticated strategies to evade the adversarial measures of WAFs. Additionally, the actor-critic architecture provides the capability to evaluate the quality of generated payloads, facilitating timely rollbacks to alleviate issues arising from the accumulation of errors.

B. Plug-in Knowledge Set and Corpus

Native LLMs can generate syntactically correct XSS payloads but lack autonomous WAF bypass retrieval. Hence, we introduce external knowledge support and a pre-defined corpus. The plug-in knowledge set is derived from descriptive texts obtained by Web crawlers that scrape methods for constructing XSS payloads from technical blogs and writeups, followed by deduplication and concise merging [16]. Unlike traditional rules, unstructured knowledge description can be directly utilized by LLMs with text comprehension capabilities, minimizing the need for excessive manual intervention. Conversely, the corpus originates from the mining of minimal invariant vocabularies in XSS payload construction, utilizing these fixed, deterministic terms as raw materials for payload synthesis to further reduce reliance on LLM capabilities.

Data Sources and Construction: Our plug-in knowledge set aggregates information from diverse sources including security researchers' blogs and writeups, academic publications from major security conferences (e.g., IEEE S&P, USENIX Security), public security repositories (GitHub, Exploit-DB), and official WAF documentation. Crucially, the knowledge set does *not* contain specific information about the CVE vulnerabilities used in our experiments, ensuring no data leakage occurs during evaluation.

Processing Pipeline: The construction involves: (1) ethical web crawling with rate limiting, (2) NLP-based extraction of technical concepts and patterns, (3) semantic deduplication using similarity clustering ($\theta = 0.85$), and (4) normalization into structured tuples $\langle$technique, context, effectiveness$\rangle$.

Knowledge Composition: The resulting knowledge set covers multiple bypass techniques, WAF-specific evasion patterns, and context-aware payload templates. Each entry includes a confidence score based on source reliability, empirical success, and expert validation factors.

Each knowledge entry includes confidence scores based on:

$$\text{Confidence} = \alpha \cdot \text{SR} + \beta \cdot \text{ES} + \gamma \cdot \text{EV} \quad (1)$$

where $\alpha + \beta + \gamma = 1$ (default: $\alpha = 0.4, \beta = 0.4, \gamma = 0.2$), SR, ES, EV refers to source_reliability, empirical_success and expert_validation, respectively.

Typical examples of the knowledge set and corpus are illustrated in Fig. 2.

{"Prerequisite": "Basic Filter", "Case": "We can hide the JS code in a malformed tag, as \<a onmouseover="alert(document.cookie)"\>xxs link\</a\> or in a image tag as <IMG """><SCRIPT>alert("XSS")</SCRIPT>"\>", "Effects": "Bypass the basic script filter.", Confidence: 42.0}

{"Prerequisite": "Keywords Filter", "Case": "We can encode the JS code in char with eval(), as Click Me!", "Effects": "Bypass the basic keywords filter.", Confidence: 54.2}

{"Prerequisite": "Keywords Filter", "Case": "We can obfuscate the JS code in unicode, as\";\u0061\u006c\u0065\u0072\u0074(1);//", "Effects": "Bypass the regex filter.", Confidence: 55.1}
... ...

(a) Knowlwdge instances

{"Function": "Alert", "Element": ["<script>", "alert(/xss/)", "</script>"]}

{"Function": "Cookies theft", "Element": ["<img src=", "link", "document.cookie", ">"]}

... ...

(b) Corpus instances

Fig. 2. Examples of knowledge set and corpus.

The introduction of the plug-in knowledge set and corpus lowers the demands on fine-tuning, facilitating the realization of a generalized prompting-based framework.

C. Tree-structured Prompting Strategy

After establishment of knowledge set and corpus for subsequent prompt engineering, it becomes essential to select applicable knowledge elements and corpus tokens based on specific scenarios and contextual information. To achieve this, a task-oriented workflow needs to be designed and constructed.

Given that, from the red teaming perspective under initial conditions, the generated XSS payloads are applied in the black-box environment [17]. Therefore, the process of generating XSS payloads can essentially be regarded as an "exploration-exploitation" process. As attempts are made, feedback is accumulated, and the prerequisites and expected effects are also acquirable along with the knowledge set, a Knowledge-based Markov Decision Process (KMDP) can be established.

Intuitive Example: Consider a red teamer testing a WAF that blocks `<script>alert(1)</script>`. A linear approach might try `<img src=x onerror=alert(1)>` next, then `<svg/onload=alert(1)>`, wasting attempts. Our tree-structured prompting instead maintains multiple

parallel hypotheses: one branch explores encoding variants (`\%3Cscript\%3E`), another explores event handlers (`onerror`), a third explores tag fragmentation. After each test, the WAF's response (block/allow) prunes failing branches and expands promising ones—similar to how a human expert mentally tracks multiple strategies simultaneously.

In contrast to the traditional Markov Decision Process (MDP) tuple $< S, A, P, R, \gamma >$, where S represents the state, A denotes the action, P refers to the transition, R stands for the reward, and γ is the discount factor. The KMDP incorporates an additional knowledge component K, resulting in the tuple $< S, A, R, \gamma, K >$. This implies that when making decisions, the LLM agent not only takes into account the current state but also combines it with a composition function of plug-in knowledge, represented as $\pi(a|S, K)$.

Implementation Details: The tree-structured prompting is implemented as a depth-first search with a maximum depth of 5 and branching factor of 3. Each node in the tree represents a specific bypass strategy (e.g., encoding, obfuscation, fragmentation), and edges represent strategy combinations. The LLM selects paths based on: (1) previous payload feedback, (2) knowledge relevance scores, and (3) corpus compatibility.

Path Selection Mechanism: At each decision point t, the LLM computes a relevance score for each knowledge element $k_i \in K$:

$$\text{score}(k_i, S_t) = \text{sim}(\text{emb}(k_i), \text{emb}(S_t)) \times w_i \tag{2}$$

where $\text{emb}(\cdot)$ denotes embedding extraction, $\text{sim}(\cdot, \cdot)$ is cosine similarity, and w_i is a learned weight from historical successes.

Corpus Integration: The corpus C provides atomic tokens $\{c_1, c_2, ..., c_n\}$. During payload construction, tokens are selected based on syntactic constraints and WAF evasion probability $P_{\text{evade}}(c_j|S_t)$, estimated from training data.

In terms of implementation, we default to using DeepSeek-V3.2 as the backend model due to its superior code generation capabilities and reasonable API cost ($0.005 per 1K input tokens, $0.015 per 1K output tokens). Alternative models were tested as shown in Section III-C and III-E, with DeepSeek-V3.2 providing the best balance of performance and cost-effectiveness.

D. Actor-critic Feedback Architecture

Actor Component: The actor LLM generates candidate XSS payloads using the tree-structured prompting strategy. For each generation attempt at step t, it produces payload p_t with associated metadata including the selected knowledge path K_{path} and corpus tokens C_{used}.

Critic Component: The critic employs a multi-faceted evaluation framework with the following criteria:

- **Syntactic Validity (Weight: 0.3):** Checks if the payload is valid JavaScript/HTML syntax using parser validation.
- **Semantic Accuracy (Weight: 0.4):** Ensures the payload maintains the intended malicious functionality using semantic analysis.
- **Bypass Potential (Weight: 0.3):** Estimates the likelihood of evading WAF detection based on pattern matching against known WAF signatures.

The critic computes a composite score:

$$\text{Score}(p_t) = \sum_{i=1}^{3} w_i \times f_i(p_t) \tag{3}$$

where $f_i(p_t)$ are normalized scores (0-1) for each criterion.

Feedback Loop: If $\text{Score}(p_t) < \tau$ (threshold $\tau = 0.7$), the critic generates specific feedback F_t identifying weaknesses:

$$F_t = \{\text{syntax\_errors}, \text{semantic\_issues}, \text{waf\_triggers}\} \tag{4}$$

Rollback Mechanism: When feedback indicates critical issues, the system performs selective rollback to the nearest viable node in the tree. The rollback distance d is determined by:

$$d = \lceil \frac{1 - \text{Score}(p_t)}{0.1} \rceil \tag{5}$$

with a maximum rollback of 3 steps to prevent excessive backtracking.

Efficiency Considerations: To balance effectiveness and efficiency, we implement two optimizations: (1) Early termination of low-probability paths based on critic predictions, and (2) Parallel evaluation of sibling nodes when possible. These reduce average tree traversal from $O(b^d)$ to approximately $O(b \log d)$ in practice.

III. Evaluation

A. Benchmark

To comprehensively evaluate the effectiveness of our generated XSS payloads, we construct a diverse and representative benchmark comprising 20 publicly available XSS vulnerabilities from real-world applications spanning multiple categories including Content Management Systems (CMS), enterprise software, web frameworks, monitoring tools, and JavaScript libraries. This diverse selection ensures our evaluation covers a wide range of XSS attack vectors (Reflected, Stored, and DOM-based) with varying complexity levels.

Table I details the benchmark composition, showing the CVE IDs, vulnerable applications, their categories, and specific XSS types. The selected CVEs range from 2017 to 2025, representing both historical vulnerabilities (for compatibility testing) and contemporary threats (for relevance testing), thus providing comprehensive temporal coverage.

Important Note: Our knowledge set and corpus contain no specific information about the 20 CVE vulnerabilities used in evaluation, eliminating data leakage concerns.

Benchmark Diversity and Coverage: Our benchmark offers superior diversity across vulnerability types (9 Reflected XSS, 8 Stored XSS, 3 DOM-based XSS), application categories (6 categories including CMS, Monitoring Tools, Web Frameworks, Enterprise Software, JavaScript Libraries, and specialized apps), temporal coverage (2017–2025), and complexity levels (from simple CMS to complex enterprise systems), comprehensively testing method adaptability.

TABLE I
BENCHMARK COMPOSITION OF 20 XSS VULNERABILITIES WITH SECURITY METRICS

Note: CVSS = Common Vulnerability Scoring System (0.0-10.0 scale). **Exploit Complexity classification**: **Low** = Basic XSS payload execution; **Medium** = Requires specific conditions or user permissions; **High** = Complex techniques or specific environment needed.

CVE ID	Vulnerable Application	Category	XSS Type	CVSS	Exploit Complexity
CVE-2017-12794	Django 1.10.7	Web Framework	Reflected	6.1	Low
CVE-2018-1000129	Jolokia 1.49	Monitoring	Stored	6.1	Medium
CVE-2019-0221	Apache Tomcat 9.0.0.M1	Application Server	Reflected	6.1	Low
CVE-2019-6341	Drupal 7.64	CMS	Stored	5.4	Medium
CVE-2019-16219	WordPress 5.0.3	CMS	Reflected	6.1	Low
CVE-2020-11022	jQuery 3.4.9	JavaScript Library	DOM-based	6.9	Medium
CVE-2021-35198	NETSCOUT nGeniusONE 6.3.0	Enterprise Software	Stored	5.4	Medium
CVE-2021-38156	Nagios XI 5.8.5	Monitoring	Reflected	5.4	Medium
CVE-2021-41728	News247 CMS 1.0	CMS	Stored	9.3	Low
CVE-2022-43144	Canteen Management System v1.0	Web Application	Reflected	5.4	Low
CVE-2023-2516	Teampass 3.0.8	Password Manager	Stored	5.4	High
CVE-2023-4174	mooSocial 3.1.6	Social Platform	Reflected	6.1	Medium
CVE-2023-29489	cPanel 11.108.0.13	Control Panel	DOM-based	6.1	Medium
CVE-2023-33829	SCM Manager 1.2	DevOps	Stored	5.4	Medium
CVE-2024-28623	RiteCMS 3.0.0	CMS	Reflected	6.1	Low
CVE-2024-30875	jquery-ui v1.13.1	UI Library	DOM-based	7.1	Low
CVE-2024-31204	Mailcow	Email System	Stored	6.1	Medium
CVE-2025-4123	Grafana 10.4.17	Monitoring	Reflected	7.6	Low
CVE-2025-45407	DiscoveryNG v6.0.8	Network Scanner	Stored	6.5	High
CVE-2025-55170	WeGIA 3.4.7	Web Application	Reflected	7.4	High

Comparison with Related Works: Our benchmark offers more comprehensive coverage than existing studies: GenXSS [7] was evaluated on a single application; HAXSS [18] tested only 10 "easy-to-crawl" vulnerabilities; and most heuristic-based tools use limited, homogeneous test sets that do not reflect production heterogeneity.

Our benchmark of 20 diverse vulnerabilities represents an 100% increase in application type diversity and 200% increase in XSS variant coverage compared to existing evaluations, providing a more realistic assessment of method generalization capabilities.

Experimental Setup: We deploy each vulnerable application in Docker containers with three security configurations: (1) without WAF protection (baseline), (2) protected by ModSecurity[1] with OWASP Core Rule Set v3.3.2 (common open-source WAF), and (3) protected by HTTPWAF[2] with default security policies (modern WAF implementation). This multi-WAF configuration allows comprehensive evaluation across varying defense strengths and rule sets. All involved vulnerable Docker configurations can be found in the VulHub[3].

We acknowledge that commercial WAFs (AWS WAF, Cloudflare) are not tested; results are lower bounds for those environments.

B. Metrics

We evaluate the quality of our generated XSS payloads primarily from the perspective of whether they can accomplish red teaming tasks within specified adversarial environments. The critical metrics encompass the executability of the generated payloads, their ability to bypass WAFs, as well as the degree of manual intervention required.

C. Experimental Results

On the established benchmark, we conducted a comparative evaluation of our proposed approach against several existing methods: the rule-based tool XSS-LOADER[4], the heuristic-based approach Chainsaw [4], the machine-learning model GAXSS [19], the reinforcement-learning based HAXSS [18], and the LLM-based method GenXSS [7]. The bypass ratio out of 20 adversarially vulnerable environments are presented in Table. II. *Note:* With 20 samples, a difference of 3 successes equals 15% absolute. Interpret as effect size, not percentage exaggeration; p-values confirm non-negligible difference.

TABLE II
EFFECTIVENESS ON DIFFERENT ADVERSARIAL ENVIRONMENT.

Setting	No WAF	ModSecurity	HTTPWAF
XSS-LOADER	16/20 (80%)	6/20 (30%)	4/20 (20%)
Chainsaw	17/20 (85%)	7/20 (35%)	4/20 (20%)
GAXSS	16/20 (80%)	10/20 (50%)	7/20 (35%)
HAXSS	17/20 (85%)	3/20 (15%)	11/20 (55%)
GenXSS	**20/20 (100%)**	13/20 (65%)	14/20 (70%)
XSSplore	**20/20 (100%)**	**19/20 (95%)**	**17/20 (85%)**

Analysis of the performance on different methods: The experimental results reveal the following insights: traditional rule-based and heuristic-based tools heavily rely on expert knowledge. When equipped with built-in WAF bypass strategies, they can counter typical open-source WAFs. However, they struggle to effectively penetrate manually-configured WAFs and often require additional human intervention. Machine-learning and reinforcement-learning based

[1]https://github.com/owasp-modsecurity/ModSecurity
[2]https://github.com/httpwaf/httpwaf2.0
[3]https://github.com/vulhub/vulhub
[4]https://github.com/capture0x/XSS-LOADER/

methods exhibit a certain degree of bypassable payloads. Nevertheless, constrained by the training dataset, achieving a high level of bypass capability remains challenging.

In contrast, our proposed *XSSplore* not only possesses relatively flexible generation capabilities but also effectively circumvents the detection of typical WAFs. By integrating knowledge and corpora into the generation process, *XSSplore* achieves higher generation validity compared to other machine-learning based methods. Moreover, with the incorporation of a tree-structured prompting and an actor-critic strategy, the XSS payloads generated by *XSSplore* are more effective and verifiable.

TABLE III
CAPACITY COMPARISON ON DIFFERENT METHOD.

Capacity	Stability	Manual	Scalability
XSS-LOADER	deterministic	require	strong
Chainsaw	deterministic	partial	strong
GAXSS	probabilistic	partial	weak
HAXSS	probabilistic	partial	weak
GenXSS	probabilistic	partial	medium
XSSplore	probabilistic	none	strong

Analysis of manual intervention: As shown in Table III, rule/heuristic methods (XSS-LOADER, Chainsaw) offer deterministic stability but require expert assistance. ML-based methods (GAXSS, HAXSS, GenXSS) are probabilistic and still need partial tuning. In contrast, *XSSplore* operates without external assistance due to its tree-structured prompting and actor-critic mechanism, while its plug-in knowledge and corpus enable easy updates and strong scalability.

D. Ablation Study

To demonstrate that the advantages of our proposed method stem from its agentic workflow rather than reliance on a specific LLM, we repeat the comparative experiments as presented in Table. IV.

TABLE IV
PERFORMANCE COMPARISON ON BASE LLMS.

Base LLM	No WAF	ModSecurity	HTTPWAF
GPT-4o	20/20 (100%)	19/20 (95%)	17/20 (85%)
Gemini-2.5	20/20 (100%)	19/20 (95%)	17/20 (85%)
Claude-3.5	20/20 (100%)	18/20 (90%)	17/20 (85%)
DeepSeek-V3.2	20/20 (100%)	19/20 (95%)	17/20 (85%)
Kimi-k1.5	20/20 (100%)	18/20 (90%)	16/20 (80%)

Analysis of different base LLMs: The experimental results demonstrate that our method exhibits stable performance on different LLMs of the same tier.

To elucidate the distinct contribution by each module within our proposed framework, we also conducted a comprehensive ablation study, as shown in Table V. Notably, the performance degradation caused by chain-structure prompting primarily stems from hallucinations.

Component Impact: Ablation analysis reveals the contribution of each component. Removing the knowledge set (-*K*) degrades performance by 10% on average (ModSecurity: -20%, HTTPWAF: -10%), confirming its importance against sophisticated WAFs; the corpus (-*C*) has a moderate impact (-8.3%), mainly reducing syntax errors. Tree-structured prompting outperforms linear prompting by 16.7%, avoiding sequential error accumulation. The actor-critic mechanism improves success rates by 10.0% on average, preventing about 65% of hallucinated payloads. Without rollback (-*R*), error accumulation reduces effectiveness by 25.0%. Overall, the three core components—knowledge base, tree-structured prompting, and actor-critic architecture—all contribute significantly. The actor-critic and rollback together show the largest impact (-35.0%), highlighting the necessity of error correction in long-chain LLM reasoning; the knowledge set and corpus provide essential domain expertise (-18.3% when both removed); and tree-structured prompting enables efficient multi-strategy exploration (+16.7% over linear).

TABLE V
PERFORMANCE COMPARISON ON DIFFERENT FRAMEWORK CONFIGURATIONS

Note: -*K* (no Knowledge), -*C* (no Corpus), *linear* (chain prompting), -*AC* (no Actor-critic), -*R* (no Rollback). **Impact** = avg. degradation over three WAF settings.

Setting	No WAF	ModSecurity	HTTPWAF	Impact
Original	20/20 (100%)	19/20 (95%)	17/20 (85%)	Baseline
-K	20/20 (100%)	15/20 (75%)	15/20 (75%)	-10%
-C	19/20 (95%)	17/20 (85%)	15/20 (75%)	-8.3%
linear	19/20 (95%)	14/20 (70%)	13/20 (65%)	-16.7%
-AC	20/20 (100%)	16/20 (80%)	14/20 (70%)	-10%
-R	18/20 (90%)	12/20 (60%)	11/20 (55%)	-25%

E. Efficiency and Cost Analysis

We analyze efficiency and cost across LLM backends on NVIDIA A100 80GB. Table VI shows DeepSeek-V3.2 achieves 9.8s generation time (33% faster than GPT-4o) and $0.021 per payload (34% savings over GPT-4o). All backends achieve 90-93% success rates; DeepSeek-V3.2 has the lowest Tokens per Success (16,506). Tree depth (3.1-3.3), branching factor (2.7-2.9), and early termination (62-68%) vary minimally across backends.

DeepSeek-V3.2 reaches 80% success in 22s (GPT-4o: 30s). With 4 concurrent workers, throughput reaches 24.5 payloads/min at 92% success. For 500 monthly payloads, DeepSeek-V3.2 costs $10.50, saving 34-45% over GPT-4o ($16.00) and Claude-3.5 ($19.00). We recommend DeepSeek-V3.2 for best performance-cost ratio, enabling 50% more tests per budget than GPT-4o.

TABLE VI
EFFICIENCY COMPARISON ACROSS LLM BACKENDS

Backend	Time (s)	Tokens	Success	Cost/Payload
GPT-4o	14.3	18,200	93.3%	$0.032
DeepSeek-V3.2	**9.8**	**15,400**	**93.3%**	**$0.021**
Claude-3.5	16.2	20,100	91.7%	$0.038
Gemini-2.5	12.7	16,800	91.7%	$0.035
Kimi-k1.5	11.5	17,300	90.0%	$0.033

IV. Limitations and Future Work

Our study presents several limitations. First, the current evaluation focuses on 20 vulnerable applications, which may not fully represent the diversity of real-world deployment scenarios. Second, while XSSplore demonstrates strong performance against common open-source WAFs, its effectiveness against proprietary or custom-configured enterprise WAFs requires further validation. Third, the reliance on LLM APIs introduces external dependencies and potential privacy concerns for sensitive environments. Fourth, the computational cost, though reasonable for security assessments, may be prohibitive for continuous monitoring applications. Fifth, the plug-in knowledge set and corpus, while effective, require periodic updates to remain relevant against evolving WAF rules. Finally, our method primarily addresses XSS vulnerabilities; extending this framework to other web security threats (e.g., SQL injection, CSRF) remains future work.

In future work, we plan to: (1) scale the benchmark to 100+ vulnerabilities; (2) evaluate on commercial WAFs (e.g., AWS WAF, Cloudflare); (3) support local deployment of open-source LLMs (e.g., Llama 3) with caching to reduce latency; (4) release an automated knowledge update pipeline; and (5) explore neuro-symbolic methods [20] that combine neural LLMs with symbolic reasoning for more interpretable and verifiable payload generation, potentially reducing hallucinations and improving generalization to unseen WAF rules.

V. Conclusion

In this paper, we propose *XSSplore*, a tree-structured prompting and actor-critic framework to automatedly generate bypassable XSS payloads for red teaming pentests. Extensive comparative experiments imply that our proposed framework significantly outperforms existing approaches, achieving high-fidelity XSS payload generation for black-box WAFs without expert intervention.

Acknowledgment

In this paper, we utilized DeepSeek-V3.2 solely for grammar refinement and stylistic polishing of pre-drafted content.

References

[1] Youkun Shi, Yuan Zhang, Tianhao Bai, Feng Xue, Jiarun Dai, Fengyu Liu, Lei Zhang, Xiapu Luo, and Min Yang. {XSSky}: Detecting {XSS} vulnerabilities through local {Path-Persistent} fuzzing. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 8255–8272, 2025.

[2] Andy Zhou, Kevin Wu, Francesco Pinto, Zhaorun Chen, Yi Zeng, Yu Yang, Shuang Yang, Sanmi Koyejo, James Zou, and Bo Li. Autoredteamer: Autonomous red teaming with lifelong attack integration. *arXiv preprint arXiv:2503.15754*, 2025.

[3] Michael C Martin and Monica S Lam. Automatic generation of xss and sql injection attacks with goal-directed model checking. In *USENIX Security symposium*, pages 31–44, 2008.

[4] Abeer Alhuzali, Birhanu Eshete, Rigel Gjomemo, and VN Venkatakrishnan. Chainsaw: Chained automated workflow-based exploit generation. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 641–652, 2016.

[5] Dennis Miczek, Divyesh Gabbireddy, and Suman Saha. Leveraging llm to strengthen ml-based cross-site scripting detection. In *Proceedings of the 2025 ACM Workshop on Wireless Security and Machine Learning*, pages 14–19, 2025.

[6] Yanan Zhang, Tingting Qiao, and Maode Ma. Fully automated xss vulnerability detection by reinforcement learning. In *International Conference on Intelligent Computing*, pages 134–144. Springer, 2025.

[7] Vahid Babaey and Arun Ravindran. Genxss: an ai-driven framework for automated detection of xss attacks in wafs. In *SoutheastCon 2025*, pages 1519–1524. IEEE, 2025.

[8] Max Landauer, Klaus Mayer, Florian Skopik, Markus Wurzenberger, and Manuel Kern. Red team redemption: A structured comparison of open-source tools for adversary emulation. In *2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 117–128. IEEE, 2024.

[9] Wenjie Jacky Mo, Qin Liu, Xiaofei Wen, Dongwon Jung, Hadi Askari, Wenxuan Zhou, Zhe Zhao, and Muhao Chen. Redcoder: Automated multi-turn red teaming for code llms. *arXiv preprint arXiv:2507.22063*, 2025.

[10] Gang Yang, Lin Ni, Shaofeng Lin, and Tao Xia. Towards few-shot llm-based vulnerability reproduction and verification for industrial web applications. In *2025 IEEE 23rd International Conference on Industrial Informatics (INDIN)*, pages 1–6, 2025.

[11] Richard Fang, Rohan Bindu, Akul Gupta, and Daniel Kang. Llm agents can autonomously exploit one-day vulnerabilities. *arXiv preprint arXiv:2404.08144*, 2024.

[12] Yuxuan Zhu, Antony Kellermann, Dylan Bowman, Philip Li, Akul Gupta, Adarsh Danda, Richard Fang, Conner Jensen, Eric Ihli, Jason Benn, et al. Cve-bench: A benchmark for ai agents' ability to exploit real-world web application vulnerabilities. *arXiv preprint arXiv:2503.17332*, 2025.

[13] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

[14] Shang-Hsuan Chiang, Tsan-Tsung Yang, Kuang-Da Wang, Wei-Yao Wang, An-Zi Yen, and Wen-Chih Peng. Tree-of-report: Table-to-text generation for sports game reports with tree-structured prompting. In *ACL 2025 Student Research Workshop*.

[15] Zhiyuan Zhang, Caixin Guo, Qichao Zhang, Xiaoxu Wu, Guogang Liao, Yongkang Wang, Xingxing Wang, and Dongbin Zhao. Llm-ac: large language models enhanced actor-critic for recommendation systems. In *International Conference on Neural Information Processing*, pages 193–205. Springer, 2024.

[16] Gang Yang, Jun He, Bo Wu, Lin Ni, Linna Fan, and Tao Xia. Enhance cve severity prediction from vulnerability description with auxiliary sentence. In *2025 28th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 2539–2544, 2025.

[17] Seyed Ali Akhavani, Bahruz Jabiyev, Ben Kallus, Cem Topcuoglu, Sergey Bratus, and Engin Kirda. Waffled: Exploiting parsing discrepancies to bypass web application firewalls. *arXiv preprint arXiv:2503.10846*, 2025.

[18] Myles Foley and Sergio Maffeis. Haxss: Hierarchical reinforcement learning for xss payload generation. In *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 147–158. IEEE, 2022.

[19] Zhonglin Liu, Yong Fang, Cheng Huang, and Yijia Xu. Gaxss: effective payload generation method to detect xss vulnerabilities based on genetic algorithm. *Security and Communication Networks*, 2022(1):2031924, 2022.

[20] Gang Yang, Lin Ni, Junfeng Geng, and Xiang Peng. Fedltn-cubesat: Neuro-symbolic federated learning for intrusion detection in leo cubesat constellations. *Mathematics*, 14(6), 2026.