A Preliminary Study of Retrieval-Augmented Generation for Sensitive Web Path Discovery in Web-based Information Systems*

Kinsey K.S. Ng†, Farah Yan, and Kevin Hung

Hong Kong Metropolitan University, Hong Kong
s1350047@live.hkmu.edu.hk, {fyan,khung}@hkmu.edu.hk

Abstract

A Retrieval-Augmented Generation (RAG) approach for security auditing in Web-based information systems is proposed to enhance traditional dictionary-based fuzzing methods. The methodology leverages public `robots.txt` files and employs an agentic RAG loop implemented with LangChain for LLM-enhanced web path discovery. The approach aims to improve the identification of sensitive paths while reducing brute-force costs and detection risk. It offers a practical method for sensitive web path discovery.

1 Introduction

1.1 Threat of Cyber Attacks on Web-based Information Systems

A web application is a software program that runs on a web server and is accessed by users through a web browser over the Internet. Web applications provide interactive services, data processing, and content delivery for a wide range of purposes, from e-commerce and digital marketplaces to social networking. Web attack is a malicious action targeting web applications with the intent to compromise their security, steal data, disrupt services, or gain unauthorized access. Web attacks leads to data leakage, financial loss, reputational damage, and disruption of e-business operations and customer experience. These attacks exploit various types of vulnerabilities in web applications.

Misconfigured access control vulnerabilities, highlighted in the OWASP Top 10 Web Application Security Risks [1] and corresponding to A01 (Broken Access Control), A05 (Security Misconfiguration), A07 (Identification and Authentication Failures), and A04 (Insecure Design), allow attackers to bypass authorization and access sensitive resources that should remain protected. Weak access control amplifies the risks associated with sensitive path discovery: unauthenticated access to administrative or backup directories can expose configuration files, database dumps, or internal APIs, while insufficient authorization on legacy endpoints can enable privilege escalation. These issues can result in breaches of confidentiality and integrity, compliance violations, and a greater blast radius for lateral movement [2].

Directory and endpoint discovery is a critical step in reconnaissance during a web attack, as attackers must first identify potentially vulnerable pages as targets. Traditional dictionary-based fuzzing approaches rely on sending large volumes of HTTP requests using extensive wordlists, which increases both network costs and the risk of detection. Existing research on

*Proceedings of The 2025 IFIP WG 8.4 International Symposium on E-Business Information Systems Evolution (EBISION 2025), Article No. 8, December 16-18, 2025, Sapporo, Japan.

†Corresponding author

enumeration strategies shows that performance depends heavily on the quality of path candidates, emphasizing the need for more efficient and targeted discovery methods [3–5]. In enterprise e-business, regular penetration testing remains essential to prevent potential attacks. The proposed RAG-based auditing workflow accelerates scoping and candidate generation, enabling more frequent and effective checks; this preparedness supports business continuity (high availability) and safeguards service integrity.

Building on retrieval-augmented generation, this study examines whether a RAG pipeline—which couples retrieval of semantically similar path hints with controlled generation—can improve discovery of sensitive paths at lower cost. Public `robots.txt` files often disclose disallowed or sensitive directories and are an underutilized source of signals [6] for more efficient identification of hidden pages in web applications. This study addresses a research questions: Does the RAG-based approach, compared to traditional wordlists, increase the efficiency of web path discovery?

1.2 Robots.txt

The `robots.txt` file is located at the root of a website (e.g., `/robots.txt`) and was originally designed to provide instructions to search engine crawlers about which areas of the site are allowed or disallowed for indexing. However, attackers often use `robots.txt` to discover sensitive resources that developers intended to hide from public view. Many of these paths are not meant to be easily found, and their exposure can lead to security issues if they reveal confidential or restricted directories.

Typical entries in `robots.txt` and their potential security implications include: `Disallow: /admin/` entries reveal the presence of administrative interfaces that should remain hidden, making them targets for unauthorized access attempts. Similarly, `Disallow: /internal-api/` paths may expose internal APIs not intended for public use, helping attackers discover undocumented or less-secured interfaces.

Beyond the basics, combining insights from `robots.txt` with reconnaissance tools (such as subdomain discovery and crawling) can reveal additional directories. Common pitfalls include enabled directory listing, exposed backup files (such as `.zip`, `.sql`, `.bak`), and weakly protected admin endpoints.

1.3 Fuzzing Techniques

Fuzzing in cybersecurity is the process that discovers accessible resources by sending crafted requests with predefined payloads, relying on a careful combination of technical dependencies, target selection, and curated wordlists or methods. In web application access control misconfiguration, practitioners merge datasets of known routing patterns, attack signatures, vulnerabilities, and endpoint lists. Path fuzzing tools, such as Gobuster, Wfuzz, and Feroxbuster, use brute-force techniques and curated path dictionaries to generate HTTP requests and analyze responses for sensitive directory discovery. While larger dictionaries can increase coverage, they also raise resource consumption and detection risk due to the increased number of requests [13, 14]. The effectiveness of path fuzzing depends on how well attackers combine their experience, tool selection, and wordlist curation for the specific target application.

1.4 Large Language Models (LLMs)

LLMs enable data synthesis and instruction generation. However, in security applications, traditional usage patterns have several limitations. Full-parameter fine-tuning demands sub-

stantial computational resources and curated labels. Prompt-only generation without retrieval increases the likelihood of hallucinations and provides inadequate coverage of domain-specific terminology. Long-context inference elevates latency and cost, while token budgets constrain how much diverse information can be included.

Embedding-model mismatch (e.g., applying general-purpose embeddings to highly specialized URL/path corpora) degrades retrieval quality, and naive or insufficient chunking reduces semantic coherence. Self-Instruct–style bootstrapping expands seed sets without manual annotation [8], but cost and drift considerations remain. Retrieval conditioning with a small language model (SLM) provides a pragmatic balance by grounding generation in relevant evidence. In multilingual sources, normalization and robust Unicode handling are required to avoid retrieval bias. These constraints motivate solutions that enable rapid processing with minimal training resources and continuous updates, such as retrieval conditioning over task-specific corpora, careful embedding selection or lightweight adaptation, and agentic RAG with LangChain for scalable coverage at lower cost.

1.5 RAG

The Retrieval-Augmented Generation (RAG) approach requires preparing a database of relevant evidence, but it is much less resource-intensive compared to fine-tuning a language model. No changes to the underlying model are needed, making it easy to update the knowledge base as new data becomes available. This flexibility allows for rapid adaptation to new domains or information without retraining. Also, the architecture supports the use of multiple agents, which can collaborate or specialize in different retrieval or generation tasks, further improving efficiency and coverage.

2 Methodology

This research introduces a pipeline for sensitive path discovery based on an agentic RAG architecture, structured into four stages: (1) Vector Database Construction, (2) Extracting Website URLs, (3) Agentic RAG Loop, and (4) Web Path Discovery (see Figure 1).

2.1 Stage 1: Vector Database Construction

Common Crawl provides extracted disallowed records from robots.txt as a large-scale source of sensitive paths [9]. A large corpus of robots.txt files is first downloaded, the Disallow entries are vectorized, and K-means clustering is applied. Based on cluster centroids and diversity, the top 10 000 frequently occurring and distinct robots.txt variants are selected for inclusion in the RAG database. Therefore,for each robots.txt file, the associated paths crawled from the source website are recorded to capture the relationship between the robots.txt file and the website.

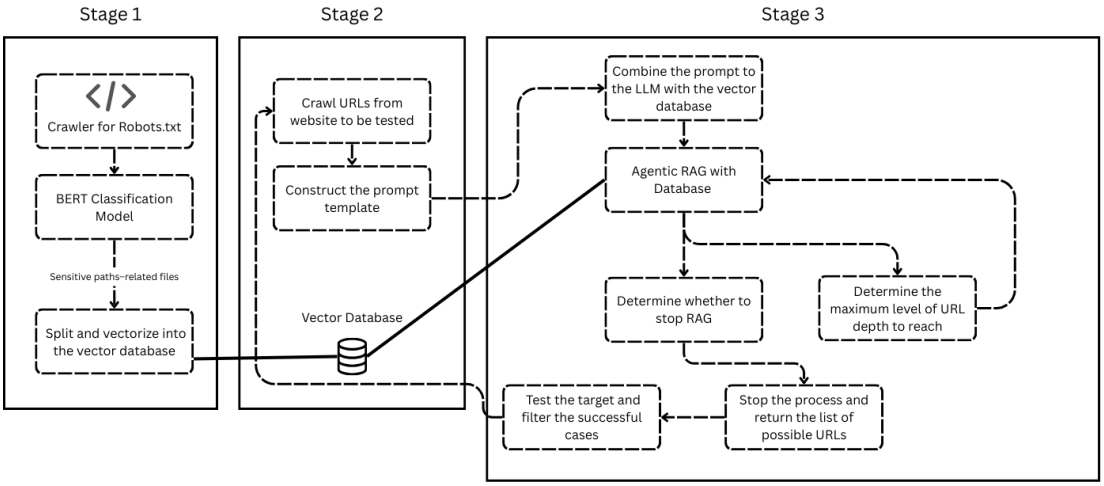

Figure 1: Overall workflow: Data acquisition and vectorization from robots.txt, target-aware prompt construction with LangChain agent coordination, and agentic RAG enumeration.

Table 1: Statistics for Robots.txt collected

Item	Value
1) Robots.txt files	3 766 879
2) Selected Distinct Robots.txt variants (for RAG)	10 000
3) Avg number of domains per top-10,000 variant	Avg: 238.85 Top: 505 610 Median: 16 Coverage: 63.41 %
4) Avg disallow URLs per robots.txt	21.08

2.2 Stage 2: Extracting Website URLs

The process begins by crawling the selected websites to collect existing paths. These discovered paths are then used to construct prompts for further prediction, combined with the site's robots.txt, which serves as a signature of the website's URLs. Only domains with detected home pages that return valid responses are considered for extraction.

2.3 Stage 3: Agentic RAG Loop

The system then enters the agentic RAG loop. Agent-style prompting can further improve grounded generation [7, 8]. The loop is implemented using the LangChain framework, which coordinates interactions among the GPT-4o-mini model, retrieval components, and validation logic in an agent-driven workflow. The process begins by constructing a prompt from the previously gathered target-site URLs and the robots.txt entries retrieved from the RAG vectorized database; these entries serve as references for sensitive keywords and pattern paths relevant to the target. Prompts use a role-based and few-shot prompting approach tailored to the web

application's structure and characteristics, considering technologies such as programming languages, frameworks, and development practices. LangChain manages both retrieval and prompt assembly and iteratively feeds newly generated URLs back into the loop until a sufficient set of candidates has been produced and stored for subsequent validation.

Upon completion, the RAG pipeline consolidates all results, producing a ranked list of suggested paths. The ranking incorporates the probability of the retrieved paths being sensitive.

2.4 Stage 4: Web Path Discovery

Generated paths are validated by issuing real HTTP requests to the originating websites from which the robots.txt files were obtained; the observed responses (e.g., HTTP 200 for valid, 404 for not found) are recorded to estimate the accuracy of the generated paths. To avoid impacting server operations, per-domain rate limits and conservative request frequencies are enforced.

3 Experimental Settings

3.1 Evaluation Datasets

For evaluation, we randomly select Web-based websites (e.g., retail, marketplaces) from downloaded robots.txt files and perform real crawling until 200 valid domains are collected (or the corpus is exhausted). A domain is considered valid if (1) its homepage returns HTTP 200 and (2) the homepage contains more than 10 distinct same-domain URLs. For each valid domain, we fetch and scrape site URLs, normalize query parameters (remove ordering, deduplicate keys, canonicalize dynamic segments), and assemble the resulting URL set as a dataset. These normalized URLs are then fed into the RAG pipeline. The pipeline generates candidate URLs by conditioning on the input URLs with prompts; the generated URLs are then used as inputs to the next iteration, and run an agentic RAG loop. In each iteration, the retriever supplies the URLs from last retrieval and query results from RAG robots.txt database, then the LLM proposes sensitive URL candidates,and keep the feedback loop in fed back to the LLM. The discovery loop terminates only when 100 unique URLs have been produced.

3.2 Baselines

Baselines include: (i) static dictionary enumeration using a breadth-first approach with a wordlist comprising the top 100 URLs extracted from disallowed entries in the RAG robots.txt database; and (ii) a prompt-only, LLM-generated wordlist for each domain, which is similar to the proposed approach but without RAG retrieval. These baselines evaluate the effectiveness of the RAG-generated URL datasets. All wordlists follow the same procedure for issuing HTTP requests, and the resulting HTTP status-code distributions are compared across methods.

3.3 Evaluation Metrics

For each of the 200 valid domains, the URLs generated by previous methods are loaded. The enumeration loop proceeds until 100 unique URLs have been produced. All generated URLs are validated by issuing real HTTP requests and collecting server responses. Status codes are mapped as follows: $TP = \{200, 403, 500\}$, $TN = \{404\}$, $FP = \{301, 302\}$, and all other codes are treated as FN for metric computation. In plain terms, an HTTP 200 (OK) indicates that the resource exists and responds successfully and is therefore counted as a true positive; 403

Table 2: Comparison across methods

	HTTP Status Codes						Confusion Matrix			
Method	200	403	500	404	301	302	Acc.	Prec.	Rec.	F1
Proposed RAG	7.27	10.19	0.54	48.72	0.00	0.00	0.667	0.668	0.465	0.496
Prompt-only LLM	2.98	5.04	0.03	20.16	0.00	0.00	0.286	0.493	0.102	0.144
Static dict. (Baseline)	0.41	0.13	0.00	2.47	0.00	0.00	0.030	0.050	0.026	0.027

Note: HTTP Status Codes fields report the average number of responses observed per domain after completing the validation test; Accuracy (Acc.), Precision (Prec.), Recall (Rec.), and F1-score (F1) are computed from the response-based classification mapping described in Section 3.

(Forbidden) indicates that the resource exists but access is denied, evidencing a valid sensitive endpoint and thus a true positive; and 500 (Internal Server Error) typically arises from an existing handler failing during processing, indicating resource existence despite a server-side failure and thus a true positive. A 404 (Not Found) indicates that the path does not resolve to a resource on the target and is counted as a true negative. Redirects such as 301/302 do not confirm the presence of the intended resource—often leading to a generic page or login—and are therefore counted as false positives for the sensitivity-detection task. Other statuses (e.g., 401, 429, 503) are treated as false negatives in summary scoring unless independent validation demonstrates resource existence.

The effectiveness of the system in identifying important URLs is assessed using following metrics [10]:

- **Code counts (200/403/500/404/301/302)**: Per-status HTTP response counts.
- **Confusion (TP/TN/FP/FN)**: Classification counts derived from the status-code mapping.
- **Accuracy (Acc.)**: Proportion of correct classifications.
- **Precision (Prec.)**: $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$.
- **Recall (Rec.)**: $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$.
- **F1-Score**: Harmonic mean of precision and recall.

4 Discussion

4.1 Benefits of Agentic RAG

Empirically (Table 2), the proposed RAG approach achieves the best overall performance, outperforming the prompt-only LLM baseline across all metrics: Accuracy improves from 0.286 to 0.667 (+0.381; +133%), Precision from 0.493 to 0.668 (+0.175; +35.6%), Recall from 0.102 to 0.465 (+0.363; +356%), and F1 from 0.144 to 0.496 (+0.352; +244%). These gains indicate that leveraging the robots.txt-grounded RAG database substantially enhances discovery effectiveness compared to LLM-only generation. Relative to traditional wordlists, the RAG approach also delivers clear improvements across the same metrics. Beyond the scores, the agent adaptively prioritizes promising candidates based on real-time feedback, reducing unnecessary brute-force attempts; and the approach avoids additional model fine-tuning while remaining practical for scalable directory fuzzing in web-based information systems.

4.2 Robots.txt

Publicly available `robots.txt` files provide reliable references for directory fuzzing, as they explicitly enumerate sensitive and restricted URL paths defined by website administrators. By incorporating these files into the RAG process, the RAG pipeline can accurately label sensitive URLs and use them as ground truth for validation. This approach enhances the accuracy of path discovery because the RAG model is guided by real-world examples rather than relying solely on generic wordlists.

4.3 Redirects and Soft-200 Responses

The status-code mapping used in this work (treating 301/302 as *FP*) reflects that redirections do not, by themselves, confirm the presence of the intended sensitive resource. In practice, however, many production stacks implement redirection logic at the application layer while returning HTTP 200, for example via HTML meta-refresh tags or JavaScript-based navigation (e.g., window.location or single-page application routing). Likewise, "soft-404" pages may return HTTP 200 with templated "not found" content. As a result, a purely status-based evaluation can misclassify client-side redirects and soft-404s as true positives.

To improve measurement accuracy without increasing request volume, lightweight content analysis can be incorporated during validation: detecting meta-refresh directives, common JavaScript redirect patterns, templated interstitials (such as login handoffs), and high-similarity "not found" layouts.

4.4 RAG Configuration and Hyperparameters

The current configuration of retrieval-augmented generation (RAG) in this work uses a set of fixed parameters and heuristics tailored to directory fuzzing and sensitive path discovery. However, mixed parameter settings may also be effective for web path enumeration. For example, two embedding channels can be considered, morphological (path structure) and contextual (semantic meaning), and combined using a static interpolation weight (α) to form a single representation [15]. Further adapting these weights dynamically based on observed novelty, hit rate, or redundancy during fuzzing could better accommodate the evolving target website. Parameters can be adjusted based on fuzzing feedback in each iteration: the agent dynamically tunes iteration thresholds and generation constraints as sensitive or novel paths emerge, modulates similarity cutoffs and the top K retrieved items according to the current hit ratio to reduce request volume relative to a constant rate, and weights sensitive term motifs to bias retrieval toward higher risk patterns under budget and rate limit constraints.

5 Conclusion

A practical, robots.txt-based workflow for directory fuzzing using RAG is presented. Leveraging public `robots.txt` with embedding and guided generation, the approach increases discovery efficiency while reducing brute-force overhead. Future work includes ablations on LLM/SLM choice, query expansion strategies, corpus composition, and broader real-world evaluations across Web-based platforms (e.g., Magento [16], Shopify [17], WooCommerce [18]).

References

[1] OWASP Foundation. Owasp top 10: 2021. OWASP Foundation, 2021. https://owasp.org/Top10/.

[2] Marc Rennhard, Malte Kushnir, Olivier Favre, Damiano Esposito, and Valentin Zahnd. Automating the detection of access control vulnerabilities in web applications. *SN Computer Science*, 3:376, 2022.

[3] Diego Antonelli, Roberta Cascella, Antonio Schiano, and Gaetano Perrone. "dirclustering": A semantic clustering approach to optimize website structure discovery during penetration testing. *Journal of Computer Virology and Hacking Techniques*, 20(4):565–577, 2024.

[4] Alberto Castagnaro, Mauro Conti, and Luca Pajola. Offensive ai: Enhancing directory brute-forcing attack with the use of language models. In *Proceedings of the 2024 Workshop on Artificial Intelligence and Security (AISec '24)*. ACM, 2024.

[5] Daniil Sigalov and Dennis Gamayunov. Dead or alive: Discovering server http endpoints in both reachable and dead client-side code. In *Journal of Information Security and Applications*, volume 82, page 103746, 2024.

[6] IETF. Robots exclusion protocol. RFC 9309, Internet Engineering Task Force, 2022. https://www.rfc-editor.org/rfc/rfc9309.

[7] Shunyu Yao et al. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[8] Yizhong Wang, Fudan Wei, et al. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*, 2023.

[9] Common Crawl Foundation. Common crawl. Open Web Corpus, 2025. https://commoncrawl.org/.

[10] Nandan Thakur, Nils Reimers, Andreas R"uckl"e, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021.

[11] Niklas Muennighoff, Nouamane Tazi, Lo"ic Magne, Nils Reimers, et al. MTEB: Massive text embedding benchmark. *arXiv preprint arXiv:2307.09677*, 2023.

[12] Jeff Johnson, Matthijs Douze, and Hervé Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

[13] I Putu Arya Dharmaadi, Elias Athanasopoulos, and Fatih Turkmen. Fuzzing frameworks for server-side web applications: a survey. *International Journal of Information Security*, 24:73, 2025.

[14] Craig Beaman, Michael Redbourne, J. Darren Mummery, and Saqib Hakak. Fuzzing vulnerability discovery techniques: Survey, challenges and future directions. *Computers & Security*, 2022.

[15] Justin Saxe and Konstantin Berlin. expose: A character-level convolutional neural network with embeddings for detecting malicious urls, file paths and registry keys. *arXiv preprint arXiv:1702.08568*, 2017. https://arxiv.org/abs/1702.08568.

[16] Adobe Inc. Adobe commerce (magento) — official site. Official website, 2025. https://business.adobe.com/products/magento/magento-commerce.html.

[17] Shopify Inc. Shopify — official site. Official website, 2025. https://www.shopify.com/.

[18] Automattic Inc. Woocommerce — official site. Official website, 2025. https://woocommerce.com/.