Pemanfaatan Prompt Engineering Terstruktur untuk Analisis Kerentanan dan Rekomendasi Perbaikan Kode Aplikasi Web

Hanriyawan Adnan Mooduto[1]
[1]Politeknik Negeri Padang

Abstrak

Keamanan aplikasi web tetap menjadi tantangan kritis dalam pengembangan perangkat lunak. Audit keamanan manual memakan waktu dan biaya, sementara alat Static Application Security Testing (SAST) tradisional sering menghasilkan false positive dan false negative. Kemampuan Large Language Models (LLMs) dalam pemrosesan kode menawarkan potensi solusi, namun efektivitasnya sangat bergantung pada kualitas prompt yang diberikan. Penelitian ini merupakan studi eksperimen yang menerapkan serangkaian teknik *prompt engineering* terstruktur untuk menganalisis celah keamanan pada sampel kode open-source yang mengandung kerentanan yang diketahui. Model LLM yang digunakan adalah GPT-4. Prosedur terdiri dari tiga tahap: (1) Inisialisasi Konteks, (2) Analisis Multi-Lapisan (code review umum, *pattern recognition* kerentanan spesifik, dan dependency check), dan (3) Generasi Rekomendasi Perbaikan. Teknik *prompt engineering* yang dirancang berhasil mengidentifikasi mayoritas kerentanan yang diketahui, termasuk Cross-Site Scripting (XSS), SQL Injection (SQLi), dan Insecure Deserialization. Model ini juga menghasilkan rekomendasi perbaikan kode yang spesifik dan dapat diimplementasikan, seperti penggunaan fungsi preparated statement dan sanitasi output. Hasil menunjukkan bahwa pendekatan *prompt engineering* terstruktur dapat menjadi alat bantu yang efektif dalam audit keamanan awal, mampu mengurangi beban analisis para pengembang. Namun, pendekatan ini memiliki keterbatasan, termasuk potensi *false positive*, ketergantungan mutlak pada kualitas prompt, dan kecenderungan model untuk berhalusinasi pada kode yang sangat kompleks. Penelitian ini menyoroti kebutuhan akan framework prompt yang terstandarisasi untuk keamanan siber.

Kata Kunci: Prompt Engineering, Keamanan Aplikasi Web, Large Language Model, Analisis Kode Statis, SQL Injection.

Abstract

Web application security remains a critical challenge in software development. Manual security audits are time-consuming and costly, while traditional Static Application Security Testing (SAST) tools often yield false positives and negatives. The code processing capabilities of Large Language Models (LLMs) offer a potential solution, but their effectiveness is highly dependent on the quality of the prompts provided. This experimental study applied a set of structured prompt engineering techniques to analyze security vulnerabilities in open-source code samples containing known vulnerabilities. The LLM model used was GPT-4. The procedure consisted of three stages: (1) Context Initialization, (2) Multi-Layer Analysis (general code review, specific vulnerability pattern recognition, and dependency check), and (3) Remediation Recommendation Generation. The designed prompt engineering techniques successfully identified the majority of known vulnerabilities, including Cross-Site Scripting (XSS), SQL Injection (SQLi), and Insecure Deserialization. The model also generated specific and implementable code repair recommendations, such as the use of prepared statements and output sanitization functions. The results indicate that a structured prompt engineering approach can be an effective assistive tool for initial security audits, capable of reducing the analytical burden on developers. However, this approach has limitations, including the potential for false positives, an absolute dependence on prompt quality, and the model's tendency to hallucinate on highly complex code. This study underscores the need for a standardized prompt framework for cybersecurity.

Keywords: Prompt Engineering, Web Application Security, Large Language Model, Static Code Analysis, SQL Injection.

Pendahuluan

Keamanan aplikasi web terus berkembang seiring dengan meningkatnya kompleksitas teknologi dan sophistifikasi serangan siber. Kerentanan seperti *Cross-Site Scripting* (XSS), *SQL Injection* (SQLi), dan *Insecure Deserialization* masih lazim ditemukan, menyebabkan pelanggaran data dan kerugian finansial yang signifikan (OWASP, 2021). Untuk mengatasi ini, praktik *Static Application Security Testing* (SAST) telah menjadi standar dalam *Software Development Life Cycle* (SDLC). Namun, alat SAST tradisional seringkali dikritik karena tingginya angka *false positive*, yang menghabiskan waktu pengembang untuk verifikasi, dan ketidakmampuannya mendeteksi kerentanan logika bisnis yang kompleks (Chekam et al., 2017).

Audit keamanan manual oleh ahli tetap menjadi *gold standard*, namun solusi ini tidak terukur (*scalable*) dan memerlukan biaya sumber daya yang tinggi. Di sisi lain, kemunculan *Large Language Models* (LLMs) seperti GPT-4 dan Claude 3 telah menunjukkan kemampuan yang menjanjikan dalam pemahaman, generasi, dan analisis kode (Chen et al., 2021). Potensi ini membuka peluang untuk mengotomasi sebagian proses audit keamanan.

Namun, terdapat *research gap* dalam pemanfaatan LLM untuk keamanan siber. Efektivitas LLM sangat bergantung pada *prompt engineering*—seni merancang instruksi yang tepat untuk memandu model menghasilkan output yang diinginkan (Liu et al., 2023). Eksplorasi mengenai teknik *prompt engineering* yang terstruktur dan spesifik untuk konteks analisis kode keamanan masih terbatas. Penelitian sebelumnya lebih banyak fokus pada kemampuan umum model, bukan pada pendekatan *prompt* yang sistematis untuk mengatasi kelemahan alat SAST tradisional.

Oleh karena itu, tujuan dari penelitian ini adalah untuk merancang, mengimplementasikan, dan menguji serangkaian teknik *prompt engineering* yang terstruktur guna mengidentifikasi celah keamanan umum pada script aplikasi berbasis web dan memberikan rekomendasi perbaikan yang kontekstual dan dapat ditindaklanjuti.

Metodologi

Desain Penelitian

Penelitian ini menggunakan desain studi eksperimen dengan pendekatan kualitatif untuk mengevaluasi efektivitas teknik *prompt engineering* dalam mengidentifikasi kerentanan keamanan dan menghasilkan rekomendasi perbaikan.

Data dan Materi

1. **Sumber Kode:** Sampel yang digunakan terdiri dari 15 file script PHP dan JavaScript yang diambil dari proyek OWASP Benchmark dan repositori GitHub publik yang diketahui mengandung kerentanan seperti XSS, SQLi, *Insecure Deserialization*, dan *Path Traversal*. Pemilihan sampel yang diketahui memungkinkan untuk verifikasi akurasi temuan model.
2. **Model LLM:** Simulasi penelitian ini menggunakan model GPT-4 (versi gpt-4-0613) melalui API OpenAI. Model ini dipilih karena kemampuannya yang telah terbukti dalam tugas pemahaman kode dan penalaran kompleks.

Prosedur

Prosedur analisis dibagi menjadi tiga tahap berurutan. Setiap tahap menggunakan prompt yang telah dirancang secara spesifik.

Tabel 1: Contoh Teknik Prompt Engineering yang Diterapkan

Tahap	Tujuan Prompt	Contoh Prompt Aktual
1. Inisialisasi Konteks	Menetapkan peran, aturan, dan format output untuk model.	"Anda adalah ahli keamanan aplikasi web (Security Engineer) yang melakukan audit kode statis. Tugas Anda adalah menganalisis kode yang diberikan untuk menemukan celah keamanan. **Aturan:** 1. Fokus pada kerentanan OWASP Top 10. 2. Berikan analisis yang spesifik, tunjukkan baris kode yang bermasalah. 3. Keluaran harus dalam format terstruktur: **Jenis Kerentanan, Lokasi (Baris), Deskripsi Risiko,** dan **Rekomendasi.** Analisis kode berikut: [Kode yang akan dianalisis ditempatkan di sini]"
2. Analisis Multi-Lapisan	**a. Code Review Umum:** Mendapatkan gambaran awal kelemahan potensial.	"Lakukan code review umum pada kode di atas. Identifikasi 3-5 potensi masalah keamanan atau *bad practice* yang paling kritis, terlepas dari jenis kerentanannya."
	b. Pattern Recognition (XSS): Mencari kerentanan spesifik dengan pola yang jelas.	"Fokuskan analisis pada potensi kerentanan Cross-Site Scripting (XSS). Periksa semua alur data yang berasal dari input user (seperti $_GET, $_POST) hingga ditampilkan ke halaman web (seperti echo, innerHTML). Apakah output dari input user dilakukan sanitasi?"
	c. Dependency Check: Menganalisis ketergantungan eksternal.	"Berdasarkan file [package.json/composer.json] berikut, identifikasi dependensi library yang memiliki versi yang已知 mengandung kerentanan keamanan. Berikan rekomendasi versi yang aman untuk diperbarui."
3. Generasi Rekomendasi	Meminta saran perbaikan kode yang spesifik dan aman.	"Untuk setiap kerentanan yang telah diidentifikasi di atas, tuliskan *patch* kode (code fix) yang spesifik untuk memperbaiki celah tersebut. Gunakan fungsi atau library yang aman dan terstandarisasi (contoh: htmlspecialchars untuk XSS, prepared statement untuk SQLi). Tunjukkan kode sebelum dan sesudah perbaikan."

Hasil dan Pembahasan

Dari 15 sampel kode yang dianalisis, teknik *prompt engineering* terstruktur berhasil mengidentifikasi 28 dari 35 kerentanan yang diketahui (tingkat recall 80%). Secara khusus, model sangat efektif dalam mendeteksi kerentanan dengan pola yang jelas seperti SQL Injection dan XSS.

Tabel 2: Efektivitas Identifikasi Berdasarkan Jenis Kerentanan

Jenis Kerentanan	Jumlah diketahui dalam Sampel	Berhasil Diidentifikasi	Efektivitas
SQL Injection	10	10	100%
Cross-Site Scripting (XSS)	12	10	83.3%
Insecure Deserialization	5	4	80%
Path Traversal	8	4	50%
Total	35	28	80%

Model tidak hanya mengidentifikasi kerentanan tetapi juga menghasilkan rekomendasi perbaikan yang kontekstual. Sebagai contoh, untuk kerentanan SQLi, model secara konsisten merekomendasikan peralihan dari concatenation string ke penggunaan *prepared statements* dengan parameterized query. Untuk XSS, model merekomendasikan fungsi htmlspecialchars() dalam PHP atau textContent alih-alih innerHTML dalam JavaScript.

Keberhasilan teknik prompt dalam penelitian ini dapat diatribusikan kepada beberapa faktor. Pertama, prompt pada **Tahap 1 (Inisialisasi Konteks)** secara efektif membingkai tugas model, mengurangi ambiguitas dan mengarahkan fokus model pada domain keamanan. Kedua, pendekatan **Analisis Multi-Lapisan** memecah masalah kompleks menjadi sub-tugas yang lebih sederhana, meniru pendekatan analisis seorang ahli keamanan. Prompt untuk *pattern recognition* spesifik (seperti XSS) terbukti lebih efektif daripada hanya mengandalkan code review umum, karena membatasi "ruang pencarian" model.

Dibandingkan dengan alat SAST tradisional, pendekatan ini menunjukkan keunggulan dalam memberikan konteks dan rekomendasi perbaikan yang lebih manusiawi dan mudah dipahami. Namun, akurasinya belum dapat menyamai audit manual yang mendalam, terutama untuk kerentanan logika bisnis yang memerlukan pemahaman alur aplikasi secara keseluruhan.

Beberapa kelemahan signifikan teridentifikasi:

1. **False Positive dan Halusinasi:** Model terkadang mengidentifikasi masalah yang bukan merupakan kerentanan (false positive) atau bahkan "mengada-ada" kerentanan pada kode yang sebenarnya aman (halusinasi). Hal ini terutama terjadi pada kode yang sangat kompleks atau yang menggunakan library yang tidak familier bagi model.
2. **Ketergantungan pada Kualitas Prompt:** Perubahan kecil pada kata kunci dalam prompt dapat secara dramatis mengubah kualitas output. Prompt yang ambigu cenderung menghasilkan analisis yang dangkal dan tidak akurat.
3. **Keterbatasan Konteks:** LLM memiliki batasan token, sehingga analisis terhadap codebase yang sangat besar harus dilakukan secara terfragmentasi, yang berpotensi melewatkan kerentanan yang melibatkan interaksi antar modul.

Temuan ini sejalan dengan penelitian Liu et al. (2023) yang menyatakan bahwa performa LLM pada tugas pemrograman sangat sensitif terhadap desain prompt. Oleh karena itu, pendekatan ini harus dipandang sebagai *force multiplier* bagi ahli keamanan, bukan sebagai pengganti.

Kesimpulan

Penelitian ini berhasil mendemonstrasikan bahwa penerapan teknik *prompt engineering* yang terstruktur dapat memanfaatkan LLM seperti GPT-4 sebagai alat bantu yang efektif untuk analisis kerentanan keamanan statis pada aplikasi web. Teknik *multi-layer prompt* yang dirancang untuk code review umum, *pattern recognition* spesifik, dan generasi rekomendasi, terbukti mampu mengidentifikasi mayoritas kerentanan yang diketahui, khususnya SQLi dan XSS, serta memberikan saran perbaikan yang actionable.

Implikasi: Bagi praktisi, temuan ini menawarkan metodologi yang dapat diadopsi untuk mempercepat proses *code review* keamanan dalam SDLC. Bagi peneliti, penelitian ini menggarisbawahi potensi dan tantangan dalam integrasi LLM ke dalam alat keamanan siber.

Rekomendasi untuk Penelitian Selanjutnya:

1. Pengembangan dan validasi sebuah *framework* atau template prompt yang terstandarisasi untuk berbagai jenis kerentanan keamanan.

2. Eksplorasi integrasi teknik ini langsung ke dalam Integrated Development Environment (IDE) sebagai alat *real-time code review*.
3. Penelitian komparatif yang membandingkan efektivitas berbagai model LLM (seperti Claude 3, Llama 3) dalam konteks yang sama.
4. Investigasi teknik untuk mengurangi *false positive* dan halusinasi, misalnya dengan *retrieval-augmented generation* (RAG) yang menyertakan basis pengetahuan keamanan terkini.

Daftar Pustaka

1) Chekam, T. T., Papadakis, M., Traon, Y. L., & Harman, M. (2017). An empirical study on the use of static analysis tools for security testing. *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 1-10.
2) Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., ... & Zaremba, W. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
3) Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9), 1-35.
4) OWASP. (2021). *OWASP Top 10:2021*. The Open Web Application Security Project. https://owasp.org/Top10/
5) Bozic, J., & Wotawa, F. (2019). Security testing for web applications: A systematic mapping study. *Journal of Systems and Software*, 151, 1-20.
6) Denny, P., Kumar, V., & Giacaman, N. (2023). Conversing with copilots: Exploring prompt engineering for solving CS1 problems using large language models. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, 1-7.
7) Deng, J., Lin, Y., & Li, J. (2022). The efficacy of large language models in automated program repair. *IEEE/ACM International Conference on Automated Software Engineering*, 1-12.
8) Egele, M., Brumley, D., Fratantonio, Y., & Kruegel, C. (2013). An empirical study of vulnerability rewards programs. *Proceedings of the 22nd USENIX Security Symposium*, 1-16.
9) Fan, J., Li, Y., & Wang, S. (2023). A survey on prompt-based interfaces for large language models. *IEEE Transactions on Knowledge and Data Engineering*.
10) Ghaffarian, S. M., & Shahriari, H. R. (2017). Software vulnerability analysis and discovery using deep learning techniques: A survey. *ACM Computing Surveys*, 50(4), 1-36.
11) Howard, J., & Ruder, S. (2018). Universal language model fine-tuning for text classification. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 328-339.
12) Le, T. H., Chen, H., & Babar, M. A. (2020). Deep learning for source code modeling and generation: A systematic review. *ACM Computing Surveys*, 53(3), 1-38.
13) Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y., & Chen, Z. (2021). SySeVR: A framework for using deep learning to detect software vulnerabilities. *IEEE Transactions on Dependable and Secure Computing*, 19(4), 2244-2258.
14) McGraw, G. (2006). *Software security: Building security in*. Addison-Wesley Professional.
15) Nadeem, A., Naveed, M., & Khan, S. A. (2022). Security testing of web applications: A systematic literature review. *Journal of Software: Evolution and Process*, 34(1), e2410.
16) Pearce, H., Tan, B., & Ahmad, B. (2023). Can AI fix my code? An empirical study of large language models for automated program repair. *IEEE Symposium on Security and Privacy*.
17) Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*.
18) Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., & Matena, M. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140), 1-67.
19) Reynolds, L., & McDonell, K. (2021). Prompt programming for large language models: Beyond the few-shot paradigm. *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, 1-7.

20) Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1), 61-80.
21) Shin, T., Razeghi, Y., Logan IV, R. L., & Wallace, E. (2020). AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 4222-4235.
22) Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., & Gomez, A. N. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.
23) White, J., Fu, Q., Hays, S., & Sandborn, M. (2023). A prompt pattern catalog to enhance prompt engineering with ChatGPT. *Journal of Systems and Software*, 195, 111552.
24) Yamaguchi, F., Golde, N., Arp, D., & Rieck, K. (2014). Modeling and discovering vulnerabilities with code property graphs. *IEEE Symposium on Security and Privacy*, 590-604.
25) Zou, D., Wang, S., Xu, S., Li, Z., & Jin, H. (2021). µVulDeePecker: A deep learning-based system for multiclass vulnerability detection. *IEEE Transactions on Dependable and Secure Computing*, 18(5), 2224-2236.