Análise Exploratória de Criação de Regras do WAF ModSecurity Utilizando Cinco LLMs Distintos

Tiago W. Morais[1], Silvio E. Quincozes[1,2], Diego Kreutz[1], Juliano F. Kazienko[3] e Vagner E. Quincozes[4]

[1]Horizon AI Labs/PPGES – Universidade Federal do Pampa (UNIPAMPA)
[2]PPGCO – Universidade Federal de Uberlândia (UFU)
[3]CTISM – Universidade Federal de Santa Maria (UFSM)
[4]IC – Universidade Federal Fluminense (UFF)

tiagomorais.aluno@unipampa.edu.br
{silvioquincozes,diegokreutz}@unipampa.edu.br
kazienko@redes.ufsm.br, vequincozes@midiacom.uff.br

Resumo. *Este trabalho apresenta uma análise exploratória da aplicação de cinco Large Language Models (LLMs) contemporâneos: OpenAI GPT-4o, DeepSeek-Chat, Google Gemini 2.5 Pro, Claude Sonnet 3.7 e Meta LLaMA 4, no processo de criação de regras para o ModSecurity. A partir da interação de dois profissionais não especialistas na ferramenta, coautores deste estudo, demonstra-se a viabilidade de gerar regras funcionais para o ModSecurity com o apoio das LLMs, por meio do refinamento iterativo de prompts, conforme evidenciado em dois cenários distintos de ataque.*

1. Introdução

O principal desafio enfrentado atualmente pelos administradores de *Web Application Firewalls*(WAFs) é a manutenção das regras, que exige ajustes constantes realizados manualmente por especialistas, consumindo um tempo considerável e demandando mão de obra altamente qualificada[Scano et al. 2024]. Trabalhos como os de [Vahid Babaey 2025] e [Babaey and Ravindran 2025] utilizam técnicas de engenharia de *prompt*, aplicadas por profissionais especializados em LLMs, para criar e refinar regras de WAFs.

No entanto, devido à escassez de especialistas em segurança cibernética [TI Inside 2025], é comum que profissionais não especializados trabalhem com WAFs [Pałka and Zachara 2011]. Nesse contexto, este estudo apresenta uma análise exploratória *ad hoc* com o objetivo de avaliar e comparar cinco LLMs modernos [eWWEK 2025]: OpenAI GPT-4o, DeepSeek-Chat, Google Gemini2.5Pro, Claude Sonnet3.7 e Meta LLaMA4, aplicados à geração de regras para o ModSecurity, quando utilizados por usuários não especializados, um tema pouco explorado na literatura relacionada.

A investigação foi conduzida por meio de refinamentos sucessivos dos *prompts* iniciais, com o objetivo de otimizar o desempenho dos LLMs [Schulhoff et al. 2024], seguidos de ajustes iterativos nas regras até que estas se tornassem eficazes contra os ataques simulados. Todas as regras geradas foram submetidas a testes em ambientes reais e controlados, simulando ataques de força bruta com *Hydra* e varreduras de vulnerabilidades com *Nmap*, com o objetivo de obter resultados mais precisos e relevantes para a pesquisa [Kraemer et al. 2004].

Ao final do experimento, foi avaliada a viabilidade do uso de LLMs na geração de regras para o ModSecurity, com uma comparação entre os modelos para identificar aquele que apresentou maior eficácia no cenário analisado.

2. Trabalhos Relacionados

A Tabela 1 resume os trabalhos relacionados que exploram o uso de LLMs na criação e refinamento de regras de WAFs. Pesquisas como [Vahid Babaey 2025] e [Babaey and Ravindran 2025] descrevem técnicas de engenharia de *prompt* que geram variações funcionais e sintaticamente válidas de ataques, aplicando-as ao OpenAI GPT-4o e Google Gemini 2.5 Pro para refinar regras de ModSecurity contra *XSS e SQLi*.

Autor	Ferramentas utilizadas	Ataques abrangidos	Domínio Aplicação	Objetivo principal da pesquisa
[Vahid Babaey 2025]	OpenAI GPT-4o Google Gemini 2.5 Pro	*XSS*	ModSecurity	Usa engenharia de *prompt* para criar e testar regras de WAF.
[Babaey and Ravindran 2025]	OpenAI GPT-4o Google Gemini 2.5 Pro	SQL-i	ModSecurity	Usa engenharia de *prompt* para criar e testar regras de WAF.
[Scano et al. 2024]	*Support Vector Machine (SVM)* *Logistic Regression (LR)* *Random Forest (RF)*	SQL-i	ModSecurity	Otimiza regras de WAF visando aumentar a taxa de detecção.
[Montaruli et al. 2024]	*Support Vector Machine (SVM)* *Logistic Regression (LR)* *Random Forest (RF)*	SQL-i	ModSecurity	Otimiza regras de WAF visando aumentar a taxa de detecção.
[Hemmati and Hadavi 2021]	*Reinforcement Learning*	Técnicas de evasão	ModSecurity e WAF-Brain	Testa regras de WAFs em busca de falhas que podem ser corrigidas.
Esta pesquisa	**OpenAI GPT-4o** **DeepSeek-Chat** **Google Gemini 2.5 Pro** **Claude Sonnet 3.7** **Meta LLaMA 4**	***Brute-Force* e *Scanner* de vulnerabilidades**	**ModSecurity**	**Análise exploratória p/ criar regras de WAF; Testa regras em ambientes reais de ataque; Verifica a eficácia de criação de regras por profissionais não especialistas em ModSecurity.**

Tabela 1. Visão geral dos trabalhos relacionados.

Outras pesquisas exploram o uso de técnicas de *Machine Learning* para otimizar regras de WAFs. [Scano et al. 2024] desenvolveram o *ModSec-Learn* para otimizar regras do ModSecurity, aumentando a eficácia em 40% e reduzindo o número de regras em 30%. Já [Montaruli et al. 2024] criaram o *ModSec-AdvLearn*, que reduziu o conjunto de regras do Modsecurity em 50% e aumentou a eficácia do WAF em 30% na detecção de ataques de *SQL Injection*. Além disso, [Hemmati and Hadavi 2021] propôs um *framework* baseado em *Reinforcement Learning* para melhorar a detecção de técnicas de evasão em WAFs.

3. Metodologia

Esta análise exploratória adota uma abordagem estruturada inspirada em [Tukey 1977] que visa investigar se profissionais com pouca ou nenhuma experiência técnica em WAFs, como o ModSecurity, são capazes de criar regras eficazes de proteção contra ataques reais utilizando LLMs. Para isso, a metodologia foi elaborada para verificar quantas iterações são necessárias até que as regras geradas se tornem funcionais, ou seja, capazes de bloquear ataques reais sem erros de execução ou prejuízos ao funcionamento do servidor Apache.

3.1. Objetivo e Processo Experimental

A análise experimental foi conduzida por dois profissionais não especialistas em ModSecurity, com níveis distintos de conhecimento técnico, ambos com noções básicas de redes e HTTP. A abordagem simula um cenário realista, no qual LLMs auxiliam equipes de

segurança cibernética que enfrentam lacunas técnicas especializadas. Esses profissionais seguiram quatro passos sequenciais no processo de geração das regras. A Figura 1 ilustra os passos do experimento:

1. **Passo-1: *Prompt* inicial**: Envio de solicitação em linguagem natural solicitando que o LLM crie uma regra com base em uma descrição do ataque. Os *prompts* refinados estão descritos na seção 3.3.
2. **Passo-2: Correção genérica**: Caso a regra gerada não funcione, um novo *prompt* é enviado com a instrução: *"A regra a seguir não protege contra o ataque; identifique o erro e corrija-a."*
3. **Passo 3: Correção com exemplo técnico**: Persistindo o problema, envia-se um *prompt* contendo dados técnicos e individuais da regra em questão ou um exemplo funcional da regra, solicitando a correção com base nessas referências.
4. **Passo-4: Correção manual**: Se a regra ainda falhar, uma análise individual é realizada e a correção é feita manualmente, com expertise humana, envolvendo ajustes de operadores, correção da sintaxe ou inclusão de parâmetros.

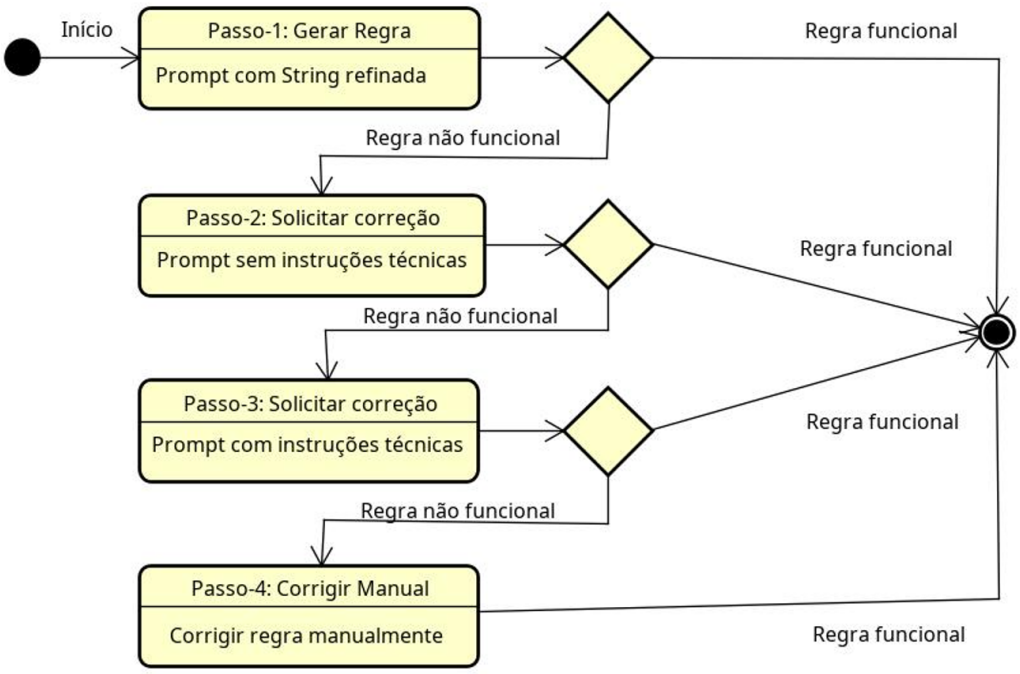

Figura 1. Passos da análise exploratória.

Durante o experimento, cada LLM foi submetido a seis *prompts* (três para o ataque *Hydra* e três para o ataque *Nmap*), refinados até atingir o critério mínimo de sucesso descrito na Seção 3.3. Esse critério foi aplicado por *prompt*, permitindo uma avaliação padronizada do desempenho dos LLMs, considerando a diversidade de respostas geradas para um mesmo *prompt*.

3.2. Ambiente de Testes e Ferramentas Utilizadas

Os testes foram realizados manualmente em um ambiente local e controlado, configurado com Apache 2.4.58 e ModSecurity, utilizando o OWASP Core Rule Set (CRS). Os cinco

LLMs foram integrados ao *Python* via APIs públicas, para avaliar a integração e compatibilidade dos LLMs com o *Python*. Ataques simulados foram realizados com *Hydra* v9.5 e *Nmap* 7.94 SVN, utilizando os seguintes comandos:

```
hydra -l usuario -P senha 127.0.0.1 http-post-form/login:
username=USER& password=PASS&:F=Login failed
nmap -sV -p 80 127.0.0.1
nmap -p 80 --script http-enum 127.0.0.1
nmap -p 80,443 --script http-security-headers 127.0.0.1
```

Todo o tráfego foi monitorado com o *Wireshark* 4.2.2 e a eficácia dos ataques e defesas foi avaliada com base na análise dos *logs* de auditoria do Apache e do ModSecurity. A defesa foi considerada bem-sucedida quando o servidor retornava erro HTTP 403 *(Forbidden)* e o ataque, quando respondido com HTTP 200 (OK). Analisando os dados capturados com *Wireshark* e os *logs* do Apache e ModSecurity, os profissionais conseguiram interpretar corretamente as respostas e confirmar se a regra cumpriu seu objetivo.

3.3. Cenários de Avaliação e Critérios de Sucesso

Dois cenários de ataque foram desenvolvidos para avaliar a capacidade dos LLMs em gerar regras funcionais: ataques de força bruta com *Hydra* e varredura de vulnerabilidades com *Nmap*. Para cada cenário, foram elaborados três *prompts* com níveis distintos de especificidade técnica, totalizando seis *prompts* de avaliação.

Durante o processo de construção, aplicou-se um refinamento iterativo aos *prompts*, com o objetivo de garantir que cada *prompt* resultasse em pelo menos 50% de regras funcionais, ou seja, (i) bloquear corretamente o ataque, (ii) não gerar falsos positivos e (iii) estar livre de erros de sintaxe. O percentual de 50% foi adotado para uniformizar a avaliação, considerando a variação na quantidade de regras retornadas pelos LLMs.

Após o refinamento, os *prompts* foram aplicados aos cinco LLMs avaliados neste estudo e todas as regras geradas foram testadas conforme descrito na Seção 3. A seguir, são apresentados os *prompts* finais utilizados no processo de geração de regras.

***Prompts* para Ataque *Hydra* (Força Bruta)**

- ***Prompt* 1 (*User-Agent* genérico)**: "Crie uma regra para ModSecurity que bloqueia tentativas de ataque *Hydra* via *User-Agent*."
- ***Prompt* 2 (Contagem de tentativas)**: "Crie uma regra para ModSecurity que incrementa o contador e bloqueia mais de 3 tentativas de ataque de força bruta por minuto."
- ***Prompt* 3 (Lista de *User-Agents* suspeitos)**: "Crie uma regra para ModSecurity que detecta *User-Agents* suspeitos, incremente o contador de IP para detectar ataques de força bruta e bloqueia a requisição quando o contador ultrapassar 3 por minuto."

Prompts* para Varredura *Nmap

- ***Prompt* 1 (*User-Agent Nmap*)**: "Crie uma regra para ModSecurity que bloqueia requisições de scanners *Nmap* com base no *User-Agent*."
- ***Prompt* 2 (Diretórios acessados)**: "Crie uma regra para ModSecurity que bloqueia o *Nmap* ao tentar visualizar diretórios no Apache."

- ***Prompt* 3 (Cabeçalhos característicos)**: "Crie uma regra para ModSecurity que bloqueia varreduras *Nmap* com base em padrões característicos de cabeçalhos HTTP utilizados por ferramentas de varredura."

Todos os *prompts* obtiveram sucesso nos cinco LLMs testados, viabilizando uma comparação padronizada do desempenho de cada modelo na geração de regras funcionais para o ModSecurity.

4. Resultados

As regras avaliadas nesta seção foram geradas com base nos seis *prompts* refinados previamente descritos na Seção 3.3, os quais foram aplicados uniformemente a todos os LLMs analisados no estudo. Os resultados a seguir apresentam o desempenho dos LLMs após a conclusão da metodologia descrita na Seção 3.

4.1. Geração de Regras para Ataques *Hydra*

Os LLMs apresentaram desempenhos variados no experimento. A classificação por desempenho foi liderada pelo OpenAI GPT-4o, que obteve 75% de sucesso no Passo-1. Em seguida, vieram o Meta LLaMA 4, com 25% no Passo-1 e 50% no Passo-3; o Google Gemini 2.5 Pro, com 16,7% no Passo-1 e 33,3% no Passo-4; o Claude Sonnet 3.7, com 14,3% nos Passos 1 e 2; e o DeepSeek-Chat, com 9,1% no Passo-1 e 18,2% no Passo-2.

Quanto às regras não funcionais, os resultados foram: 25% para o OpenAI GPT-4o; 25% para o Meta LLaMA 4; 50% para o Google Gemini 2.5 Pro; 71,4% para o Claude Sonnet 3.7; e 72,7% para o DeepSeek-Chat.

4.2. Geração de Regras para Varredura *Nmap*

Os LLMs também apresentaram desempenhos variados nesse cenário. A classificação por desempenho foi liderada pelo Google Gemini 2.5 Pro, com 100% das regras atendendo ao cenário de sucesso concentradas no Passo-1, seguido pelo Meta LLaMA 4, com 66,7% no Passo-1 e 33,3% no Passo-2; pelo OpenAI GPT-4o, com 57,1% de sucesso no Passo-1; pelo DeepSeek-Chat, com 42,9% no Passo-1; e pelo Claude Sonnet 3.7, com 33,3% no Passo-1 e 16,7% no Passo-2.

Quanto às regras não funcionais, os resultados foram: 42,9% para o OpenAI GPT-4o; 50% para o Claude Sonnet 3.7; 57,1% para o DeepSeek-Chat; enquanto o Meta LLaMA 4 e o Google Gemini 2.5 Pro não apresentaram regras não funcionais.

4.3. Desempenho geral dos LLMs

O desempenho geral dos LLMs foi classificado não pela quantidade de regras geradas, mas pelo percentual distribuído entre os passos descritos na seção 3.3. O OpenAI GPT-4o apresentou os melhores resultados, com 63,64% das regras geradas bloqueando os ataques no Passo-1 do experimento, atendendo assim ao cenário de sucesso proposto.

O Meta LLaMA 4 ocupou uma posição destacada, demonstrando capacidade adaptativa ao bloquear aproximadamente 50% dos ataques no Passo-1, 16,67% no Passo-2 e 33,33% no Passo-3 do experimento, evidenciando que o LLM realizou ajustes progressivos para aprimorar a eficácia das regras geradas.

O DeepSeek-Chat e o Claude Sonnet 3.7 apresentaram desempenho intermediário semelhante, com geração de regras eficazes nos Passos 1 e 2. Observou-se, porém, uma leve vantagem do DeepSeek no Passo-1 (20% contra 18,18% do Claude Sonnet 3.7), enquanto o Claude se destacou no Passo-2 (13,64% contra 10% do DeepSeek).

Já o Google Gemini 2.5 Pro foi o LLM com menor desempenho entre os testados, embora tenha gerado 40% das regras eficazes no Passo-1, produziu 40% de regras não funcionais e os demais 20% dependeram de correção manual no Passo-4 do experimento, exigindo intervenção individualizada com *expertise* humana.

Em síntese, os resultados revelam uma hierarquia de desempenho clara: OpenAI GPT-4o (63,64% de eficácia), Meta LLaMA 4 (melhora incremental na qualidade das regras ao longo das iterações), DeepSeek-Chat e Claude Sonnet 3.7 (desempenho intermediário equilibrado) e Google Gemini 2.5 Pro (menor autonomia na geração de regras funcionais).

A diferença na complexidade dos ataques também se mostrou determinante, com os cenários *Hydra* exigindo três ajustes nos *prompts*, em contraste com nenhum ajuste necessário para o ataque *Nmap*. A Figura 2 apresenta o desempenho geral dos LLMs.

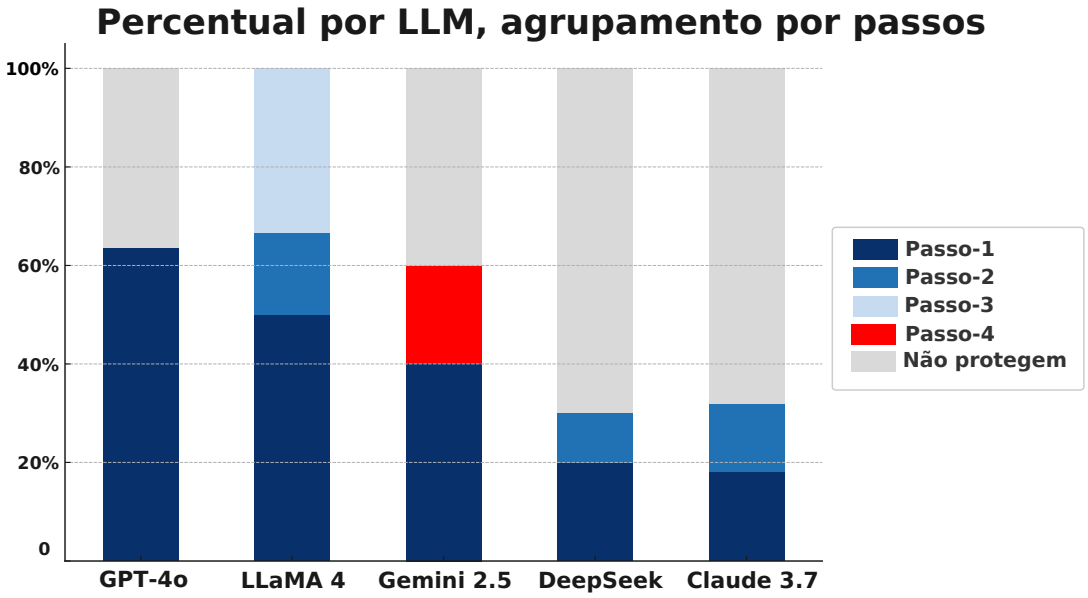

Figura 2. Desempenho geral na geração de regras de ModSecurity.

No que diz respeito às regras geradas que não protegeram contra os ataques propostos, o DeepSeek-Chat apresentou os maiores resultados, com 70% de regras não funcionais, seguido pelo Claude Sonnet 3.7, com 68,18%, pelo Google Gemini 2.5 Pro, com 40% e pelo OpenAI GPT-4o, com 36,36%.

Já o Meta LLaMA 4 gerou o menor número de regras no experimento; todas as regras protegeram contra os ataques, ajustadas nos passos 1, 2 e 3 do estudo. Esses resultados evidenciam variações importantes entre os LLMs, tanto em desempenho quanto na quantidade de regras geradas.

4.4. Consistência dos *prompts* entre execuções

Além da eficácia, avaliou-se a consistência das respostas dos LLMs mediante *prompts* idênticos submetidos repetidamente. Os seis *prompts* refinados da seção 3.3 foram submetidos três vezes ao mesmo LLM para verificar a consistência das regras geradas.

Os LLMs exibiram comportamentos distintos e variações nas respostas, mesmo com *prompts* idênticos. O OpenAI GPT-4o gerou regras logicamente semelhantes, variando mensagens, IDs e expressões regulares. O Claude Sonnet 3.7 manteve estruturas similares, alterando apenas identificadores. O DeepSeek-Chat aumentou a complexidade das regras sem solicitação explícita. O Google Gemini 2.5 Pro produziu respostas repetitivas com pequenas variações sintáticas. O LLaMA 4 apresentou as maiores variações na estrutura e lógica das respostas.

As diferenças entre os LLMs podem ser ilustradas pelas regras geradas pelo OpenAI GPT-4o e pelo Google Gemini 2.5 Pro para bloquear varreduras *Nmap* baseadas no *User-Agent*, conforme apresentado a seguir:

```
SecRule REQUEST_HEADERS:User-Agent "@contains Nmap" "id:10001,phase:1,
   deny,status:403,msg:'Nmap Scan Detected'"

SecRule REQUEST_HEADERS:User-Agent "Nmap", "id:20002, phase:2, log,
   deny, msg:'Scan Nmap detectado'"
```

Um exemplo de transformação da regra ao longo do experimento foi o *Prompt Hydra*-3 no Google Gemini 2.5 Pro. No Passo-1, as regras geradas não bloquearam o ataque. No Passo-2, o *prompt* foi reescrito solicitando correção, resultando em uma regra mais específica, mas ainda ineficaz. No Passo-3, mesmo com a inclusão de exemplos técnicos no *prompt*, a regra permaneceu não funcional. A funcionalidade só foi alcançada no Passo-4, após correção manual da regra.

Os resultados obtidos demonstram que os LLMs podem adotar estilos diferentes de construção de regras, com variações na fase de execução, sintaxe e mensagens de *log*. Isso destaca a importância de compreender essas diferenças para melhorar a eficácia na geração de regras de ModSecurity.

5. Conclusão e Trabalhos Futuros

Os LLMs demonstraram bom desempenho inicial na criação de regras para WAFs, mas, em alguns casos, exigiram supervisão humana para ajustes específicos, como correções de sintaxe e a inclusão de operadores `regex`. A análise revelou que, com supervisão adequada, os LLMs podem automatizar o processo de criação e ajuste de regras para ModSecurity, tornando-o mais ágil e eficiente. Com instruções apropriadas, os modelos demonstraram precisão na geração dessas regras.

Trabalhos futuros visam expandir o experimento para incluir ataques evasivos emergentes, como *SQL Injection*, *XSS* e *DDoS*, além de analisar qualitativamente as regras geradas por diferentes LLMs a partir de um mesmo *prompt*. A interação entre as regras e o impacto da ordem dessas regras no comportamento do ModSecurity também serão avaliados, considerando a influência da precedência na resposta do WAF em ambientes reais. Em resumo, os resultados indicam que os LLMs, aliados à expertise humana, oferecem uma solução prática e eficaz para automatizar a criação de regras de ModSecurity.

Agradecimentos. Esta pesquisa contou com apoio parcial da CAPES, código de financiamento 001; da RNP, por meio do Programa Hackers do Bem e do GT LFI – *Learn From Incidents*; e da FAPERGS, por meio dos termos de outorga 24/2551-0001368-7 e 24/2551-0000726-1.

Referências

Babaey, V. and Ravindran, A. (2025). Gensqli: A generative artificial intelligence framework for automatically securing web application firewalls against structured query language injection attacks. *Future Internet*, 17(1).

eWWEK (2025). 9 Melhores Modelos de Linguagem Ampla (2025) para sua Pilha de Tecnologia. https://www.eweek.com/artificial-intelligence/best-large-language-models/. Acessado em 29 de Julho de 2025.

Hemmati, M. and Hadavi, M. A. (2021). Using deep reinforcement learning to evade web application firewalls. In *18th ISCISC*, pages 35–41. IEEE.

Kraemer, S., Carayon, P., and Duggan, R. (2004). Red team performance for improved computer security. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 48, pages 1605–1609.

Montaruli, B., Floris, G., Scano, C., Demetrio, L., Valenza, A., Compagna, L., Ariu, D., Piras, L., Balzarotti, D., and Biggio, B. (2024). Modsec-advlearn: Countering adversarial sql injections with robust machine learning.

Pałka, D. and Zachara, M. (2011). Learning web application firewall-benefits and caveats. In *International Conference on Availability, Reliability, and Security*. Springer.

Scano, C., Floris, G., Montaruli, B., Demetrio, L., Valenza, A., Compagna, L., Ariu, D., Piras, L., Balzarotti, D., and Biggio, B. (2024). Modsec-learn: Boosting modsecurity with machine learning. In *International Symposium on Distributed Computing and Artificial Intelligence*, pages 23–33. Springer.

Schulhoff, S., Ilie, M., Balepur, N., Kahadze, K., Liu, A., Si, C., Li, Y., Gupta, A., Han, H., Schulhoff, S., et al. (2024). The prompt report: a systematic survey of prompt engineering techniques. *arXiv preprint arXiv:2406.06608*.

TI Inside (2025). Investimentos em cibersegurança no brasil. Relatório técnico, TI Inside Research. Acessado em: 16 mai. 2025.

Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison-Wesley, Reading, Massachusetts.

Vahid Babaey, A. R. (2025). GenXSS: an AI-Driven Framework for Automated Detection of XSS Attacks in WAFs. `https://arxiv.org/html/2504.08176v1`.