Segurança e governança de código

Pense no LLM como **um estagiário superpoderoso** que:

- escreve código com velocidade,
- tem acesso a ferramentas (executar scripts, chamar APIs),
- e “aprende” com o que você alimenta (RAG, embeddings, fine-tuning).

Agora, dê a esse estagiário:

(a) acesso a repositórios e segredos

(b) autonomia para executar tarefas

(c) dados sensíveis de clientes.

Sem controles, você criou o cenário perfeito para: **vazamentos, corrupção de código, custos explosivos e ações irreversíveis**.

É por isso que segurança e governança deixam de ser “camada final” e viram **pré-requisito**.

> **Moral da história:** com IA, o erro “pequeno” escala muito rápido — porque acontece mais cedo (no dev), mais fundo (no dado) e mais amplo (em várias ferramentas ao mesmo tempo).

Felizmente, uma comunidade já investigou estes (e outros) potenciais cenários que exigem segurança e governança: a OWASP.

A OWASP é uma comunidade global de segurança de software, famosa pelo “Top 10” de riscos para aplicações web (aquele documento que todo curso de segurança cita). A ideia é simples: criar **listas curtas, consensuais e práticas** do que mais causa incidentes — e como reduzir.

Com a popularização de LLMs e GenAI, a OWASP criou o **GenAI Security Project**: um hub com guias, checklists e o **Top 10 para aplicações com LLMs**. O conteúdo fica em **genai.owasp.org** e organiza, para 2025, os **10 principais riscos** de produtos que usam IA generativa. É um esforço comunitário, atualizado a partir da versão 2023/24, com explicações, exemplos e mitigação.

Segue um resumo sobre a lista:

1. **Injeção por prompt**
   Texto malicioso (até invisível) “reprograma” o modelo: ignora regras, vaza dados, aciona funções indevidas. Pense em um diff de PR com instruções escondidas que o bot de revisão obedece.

2. **Vazamento de informação sensível**
   PII, segredos, dados internos e até política de segurança escorrem em respostas, logs, embeddings ou no próprio prompt do sistema.

3. **Cadeia de suprimentos de IA**
   Modelos, LoRA/adapters e datasets de terceiros chegam com viés, backdoor ou malware — e você “assina” isso no seu build.

4. **Envenenamento de dados/modelos**
   Dados de treino/fine-tuning/embedding manipulados instalam gatilhos (backdoors) ou distorcem comportamento.

5. **Manuseio inseguro da saída**
   Tratar saída de LLM como “confiável” e jogar direto em HTML/SQL/shell: dá XSS, SQLi, RCE e afins.

6. **Agência excessiva (agentes soltos)**
   Ferramentas demais, permissões demais, autonomia demais → ações destrutivas sem freio humano.

7. **Vazamento do prompt de sistema**
   Prompt não é cofre: se ele contém segredos ou define autorização, o atacante descobre e explora.

8. **Falhas em vetores/embeddings (RAG)**
   Vector store sem ACL/partição vaza entre times; dá até para **reconstruir** partes do texto a partir do embedding (inversão).

9. **Desinformação (alucinação + overreliance)**
   A IA fala bonito, mas inventa pacote/biblioteca; dev confia e traz vulnerabilidade para o código.

10. **Consumo sem limites (custo/DoS/extração)**
    Inputs gigantes, requisições em massa e exposição de probabilidades drenam recursos, explodem a conta e facilitam extrair o comportamento do modelo.

A lista completa, com páginas individuais e mitigações, está reunida no **LLM Top 10 (2025)**. Confira em: https://genai.owasp.org/