GUIA GRÁTIS · BYLUNADEV

Os 7 Furos

O teste de 15 minutos para saber se o seu app feito com IA está vazando dinheiro

1 2 3 4 5 6 7

bylunadev

Antes de começar

Você criou um app com IA. Ele funciona. A pergunta deste guia é outra: **ele aguenta alguém tentando quebrar?**

Apps feitos com IA caem sempre nos mesmos sete furos. A IA otimiza para *funcionar*, e segurança é justamente o que não aparece quando tudo funciona: um app seguro e um furado são idênticos na tela.

São **sete testes** de cerca de dois minutos cada, para copiar e colar. No fim, você soma os pontos e descobre a sua nota.

Como usar

Faça na ordem e marque **Passou** ou **Falhou**. Abra o terminal (no Mac: app **Terminal**; no Windows: **Git Bash** ou **WSL**) e cole estas linhas uma vez, trocando só o endereço do app. Elas guardam o endereço (`$APP`) e o formato das chaves secretas (`$CHAVE`) que os testes usam, então você não precisa editar mais nada. Se fechar o terminal, cole de novo.

```
APP=https://seu-app.com
CHAVE='sk-[A-Za-z0-9_-]{20,}|sk_live_[A-Za-z0-9]{20,}'
CHAVE="$CHAVE|rk_live_[A-Za-z0-9]{20,}|gsk_[A-Za-z0-9]{20,}"
CHAVE="$CHAVE|AIza[0-9A-Za-z_-]{35}"
```

Só no que é seu

Rode estes testes **apenas no seu próprio app**, ou em um app cujo dono autorizou por escrito. Testar o sistema de outra pessoa sem autorização é crime (Brasil: art. 154-A do Código Penal; Argentina: art. 153 bis do Código Penal).

O princípio que explica os sete furos

> **O navegador é território inimigo.**

Tudo o que chega no navegador (código, variáveis, respostas) pode ser lido e alterado pelo usuário. Por isso, segredos **nunca** vão para o front, quem decide o que cada um pode fazer é **o servidor**, e todo dado que vem do navegador é suspeito até o servidor conferir. Os sete furos são variações dessa ideia.

FURO 1

A chave da API está no front

O que é. Para o app falar com a IA (OpenAI, Anthropic, Gemini), ele usa uma chave secreta. Se a chamada é feita direto do navegador, a chave vai junto, dentro do JavaScript que qualquer visitante baixa.

Por que é grave. Existem robôs que varrem a internet atrás de chaves expostas. Quando acham uma, usam a sua conta até o limite do cartão. É assim que nasce a fatura de milhares de dólares.

Teste (2 minutos). Primeiro, de fora, como um robô veria o seu site:

```
curl -sL "$APP" | grep -oE '(src|href)="[^"]+\.js[^"]*"' \
  | cut -d'"' -f2 | sort -u | while read js; do
      case "$js" in http*) echo "$js";; /*) echo "$APP$js";;
                    *) echo "$APP/$js";; esac
    done | xargs -n1 curl -sL | grep -oE "$CHAVE" | sort -u
```

Esse comando só enxerga os arquivos que a página inicial carrega. Para pegar **todos** (inclusive os que só abrem em outras páginas), rode também, dentro da pasta do projeto, depois de `npm run build`:

```
grep -rhoE "$CHAVE" dist build out .next/static .output/public \
  2>/dev/null | sort -u
```

Sem terminal? Abra o app no Chrome, **navegue pelas páginas** (para o navegador carregar tudo), aperte **F12**, vá em **Sources**, aperte **Ctrl+Shift+F** (no Mac, **Cmd+Option+F**) e pesquise `sk-`, `sk_live` e `AIza`.

Resultado	Significado
Nada apareceu	✅ **Passou**
Apareceu algo parecido com uma chave (`sk-…`, `sk_live_…`, `AIza…`)	❌ **Falhou**: troque a chave **hoje** no painel do provedor

Por que a chave foi parar lá

Variáveis com estes prefixos **vão para o navegador de propósito**: `NEXT_PUBLIC_`, `VITE_`, `EXPO_PUBLIC_`, `REACT_APP_`, `PUBLIC_`. É o erro mais comum de app feito com IA. Abra o seu `.env` e confira: nenhuma variável com esses prefixos pode guardar segredo. Usa **Supabase**? A chave `anon` (ou *publishable*) pode ficar no front, ela foi feita para isso. A `service_role` (ou *secret*) **nunca**: ela ignora todas as regras de acesso do banco.

Chave do Google (AIza...)

Algumas chaves `AIza` são públicas por natureza, como a do Firebase e a do Google Maps, e aparecem no front em qualquer app. Elas só são problema se não tiverem restrição. Se a chave achada for a do **Gemini**, é falha. Na dúvida, abra o *Google Cloud → APIs e serviços → Credenciais* e veja a que ela dá acesso.

Se falhou

Apagar a chave do código não basta: ela já foi copiada. **Revogue** a chave no painel do provedor e gere outra. Depois, mude a arquitetura: o navegador chama o **seu** servidor, e só o servidor chama a IA. O *App Blindado* mostra o código desse "proxy" passo a passo.

FURO 2

O `.env` subiu para o Git

O que é. O `.env` guarda as senhas do app: chaves de API, banco, segredo do login. Se ele foi enviado ao GitHub, mesmo que uma vez só e depois apagado, continua no **histórico**.

Por que é grave. Repositório público: qualquer um lê. Privado: qualquer pessoa que um dia tiver acesso (um freelancer, um app conectado) lê também.

Teste (2 minutos). Dentro da pasta do projeto, rode os três:

1. Algum `.env` está sendo enviado agora?

```
git ls-files | grep -iE '(^|/)\.env' \
  | grep -viE '\.(example|sample|template)$'
```

2. Algum `.env` já foi enviado um dia, em qualquer pasta?

```
git log --all --format= --name-only \
  -- '.env' '.env.*' '*.env' '*.env.*' \
  | grep . | grep -viE '\.(example|sample|template)$' | sort -u
```

3. Alguma chave ficou escrita no código, em qualquer versão antiga?

```
git log -p --all | grep -oE "$CHAVE" | sort -u
```

E confira se o repositório é público: abra o endereço dele em uma **janela anônima**. Se abrir, é público.

Resultado	Significado
Os três comandos não mostram nada	✅ **Passou**
Aparece algum arquivo `.env` ou alguma chave	❌ **Falhou**
Repositório público **e** algum `.env` no histórico	🚨 **Emergência**: troque todas as senhas agora

Se falhou

Troque **todas** as chaves do arquivo, ponha `.env*` no `.gitignore` e guarde os segredos nas variáveis secretas da hospedagem (Vercel, Cloudflare, Railway...). Limpar o histórico só ajuda **depois** de trocar as senhas, nunca no lugar disso.

FURO 3

A IA do app funciona sem login

O que é. Seu app tem uma função de IA (um chat, um corretor, um gerador). Se a rota que chama a IA não confere quem está pedindo, qualquer pessoa na internet usa a sua IA, e você paga.

Teste (2 minutos).

1. Abra o app logado, aperte **F12** e vá em **Network**.
2. Use a função de IA uma vez. Aparece uma linha nova (por exemplo, `chat` ou `generate`). Clique nela com o botão direito e escolha **Copy → Copy as cURL (bash)**.
3. Cole no terminal, mas **antes de apertar Enter, apague as linhas que levam a sua identidade**: as que começam com `-H 'cookie:`, `-H 'authorization:` ou `-b`. Assim o pedido chega como se fosse de um desconhecido.
4. No fim do comando, acrescente `-s -o /dev/null -w '%{http_code}\n'` e aperte Enter.

Resultado	Significado
`401`, `403`, ou um redirecionamento para o login (`302`, `307`)	✅ **Passou**
`200`	❌ **Falhou**: qualquer um usa a sua IA de graça
Outro número (`400`, `500`...)	⚠️ Inconclusivo: confira se apagou só as linhas de identidade e repita

Se falhou

A rota precisa conferir a sessão **no servidor** antes de chamar a IA, e responder `401` quando não houver usuário. Esconder o botão na tela não protege nada: o ataque não passa pela tela.

FURO 4

Não existe limite por usuário

O que é. Mesmo com login, um único usuário (ou um robô com uma conta) pode chamar a IA mil vezes por minuto. Sem limite por usuário, uma conta grátis vira uma torneira aberta na sua fatura.

Teste (2 minutos). Agora o contrário do furo 3: copie de novo a chamada da IA (**Copy → Copy as cURL (bash)**), desta vez **sem apagar nada**, e cole dentro deste laço:

```
for i in $(seq 1 40); do
  COLE_AQUI -s -o /dev/null -w '%{http_code} '
done; echo
```

Troque `COLE_AQUI` pelo comando copiado. Ele pode ter várias linhas, tudo bem: o importante é que o `-s -o /dev/null…` fique logo depois da última.

Resultado	Significado
Em algum momento aparece `429`	✅ **Passou**: existe limite
40 vezes `200`	❌ **Falhou**, a não ser que você tenha programado um limite maior que 40 de propósito

Cuidado com este teste

Cada chamada que passar é uma chamada real e paga à IA. Quarenta costuma custar centavos, mas não aumente o número.

Se falhou

Guarde no banco quantas vezes cada usuário usou a IA **no dia** e recuse acima do limite (com `429`). Melhor ainda: um **teto total do app por dia**, para que nem mil contas juntas passem do seu orçamento.

FURO 5

Não há limite de gasto no provedor

O que é. Mesmo com tudo certo no código, um erro ou um abuso pode gastar mais do que você imagina. A última linha de defesa é o **limite de gasto no painel do provedor de IA**.

Teste (2 minutos). Abra o painel do seu provedor e procure:

Provedor	Onde procurar
OpenAI	*Settings* → *Billing*: se a conta usa créditos pré-pagos, veja se a **recarga automática** está ligada. Em *Settings* → *Limits*, orçamento e alertas por e-mail
Anthropic	*Console* → *Settings* → *Billing* (recarga automática) e *Settings* → *Limits* (limite de gasto mensal)
Google (Gemini / Cloud)	*Billing* → *Budgets & alerts*. Atenção: o orçamento do Google **só avisa, não bloqueia**
Cloudflare Workers AI	*Billing*: sem cartão cadastrado, a cota grátis simplesmente acaba; com plano pago, configure *Notifications*

Resultado	Significado
Existe limite **e** alerta, com valor que você aguenta pagar, e a recarga automática está desligada (ou tem teto)	✅ **Passou**
Não existe, ou você não sabe onde fica	❌ **Falhou**

Se falhou

Configure hoje um limite baixo (por exemplo, o dobro do que você gastou no mês passado) e um alerta em 50%. Em conta pré-paga, o crédito **é** o limite: deixe pouco saldo e desligue a recarga automática. Os painéis mudam de lugar com o tempo: se não achar o menu, procure por *budget*, *usage limit* ou *spend limit* na ajuda do provedor.

FURO 6

Arquivos secretos abertos no site

O que é. Às vezes o `.env`, a pasta `.git` ou um backup acabam publicados junto com o site. Robôs testam esses endereços o dia inteiro.

Teste (1 minuto).

```
for p in /.env /.env.local /.env.production /.env.development \
         /.git/config /.git/HEAD /backup.zip /db.sql /dump.sql \
         /.DS_Store /config.json; do
  code=$(curl -s -o /dev/null -w '%{http_code}' "$APP$p")
  printf '%-18s %s\n' "$p" "$code"
done
```

Resultado	Significado
Tudo `404` (ou `403`)	✅ **Passou**
Algum `200`	❌ **Falhou**: abra o endereço e veja o que está exposto

Pegadinha

Alguns sites respondem `200` para **qualquer** endereço e mostram a página inicial. Se deu `200`, abra o endereço no navegador: se aparecer a sua home, é falso alarme; se aparecer texto com senhas, é real.

Se falhou

Tire o arquivo do servidor, **troque tudo** o que estava nele e ajuste o deploy para publicar só a pasta de build (`dist`, `out`, `.next`...), nunca a raiz do projeto.

FURO 7

Você pediu para a IA *revisar*, não para *atacar*

O que é. "Revise a segurança do meu app" gera uma resposta educada que quase sempre termina em "está tudo certo". A IA lê o código com o mesmo olhar de quem o escreveu. Para achar furos, ela precisa trocar de lado.

Teste (3 minutos para colar, o resto a IA faz). Abra o seu assistente de código (Claude Code, Cursor...) **dentro do projeto** e cole:

```
Você é um pentester contratado para invadir ESTE app, com autorização
do dono. Não revise: ATAQUE. Para cada rota da API e cada página
protegida:
1. tente usar sem login;
2. tente acessar dados de OUTRO usuário trocando ids;
3. tente virar admin ou pular o pagamento;
4. procure segredos no código que vai para o navegador (inclusive
   variáveis NEXT_PUBLIC_, VITE_ e parecidas) e no histórico do git;
5. tente gastar a IA sem limite;
6. se o app usa Supabase ou Firebase, confira se as regras do banco
   (RLS / Security Rules) impedem ler e alterar dados de outro usuário.
Para cada tentativa, mostre o comando exato, o resultado esperado se
estiver seguro e se você acha que está vulnerável. Não altere nenhum
arquivo. No fim, liste os achados por gravidade.
```

Resultado	Significado
Você já fez isso e corrigiu o que apareceu	✅ **Passou**
Nunca fez, ou só pediu "revise"	❌ **Falhou**

Dica de quem faz isso de verdade

A IA também erra quando ataca. Ela pode dizer que algo é vulnerável quando não é, e vice-versa. **Confirme cada achado rodando o comando você mesmo** antes de acreditar.

A sua nota

Some **1 ponto por teste que passou**.

Pontos	Diagnóstico
7	🟢 **Base sólida.** Você fechou os furos que derrubam a maioria dos apps. O próximo nível é autorização (um usuário vendo dados de outro), painel de admin, força bruta no login e dependências.
5–6	🟡 **Exposto.** Um ou dois furos podem custar caro. Corrija os que falharam esta semana, começando pelos furos 1, 2 e 3.
3–4	🟠 **Vulnerável.** É questão de tempo até um robô achar. Pare de lançar funcionalidades e corrija antes.
0–2	🔴 **Emergência.** Troque as chaves hoje e coloque um limite de gasto agora, antes de qualquer outra coisa.

Passar nos 7 não é estar seguro

Estes testes pegam os furos mais **comuns e caros**. Eles não cobrem tudo. Um app pode passar nos sete e ainda deixar um usuário ver os dados de outro trocando um número na URL, que é o furo mais comum *depois* destes.

O próximo passo

Este guia mostrou **onde** estão os furos. Fechar cada um, e achar os que não cabem em um teste de dois minutos, é o assunto do **App Blindado**.

App Blindado · Guia técnico de cibersegurança para quem cria apps com IA

- O código da correção de cada um dos 7 furos, mais os que vêm depois deles.
- Autenticação e autorização: força bruta, sessão, painel de admin e o furo de "trocar o id na URL".
- Teto de gasto da IA no código, à prova de duas requisições ao mesmo tempo.
- Injeção, XSS, CORS, webhooks e cabeçalhos de segurança.
- O roteiro completo para a IA atacar o seu app, com prompts por área.
- O caso real de uma plataforma auditada, com cada ataque e o resultado.
- Um checklist de 47 itens com **Nota de Segurança** de 0 a 100.

E se você prefere que alguém faça por você: **eu faço a auditoria do seu app**, com contrato, ataque autorizado, relatório e reteste depois das correções.

> **Fale comigo:** @bylunadev