Web Security Audit na Prática

Guia pt-BR: do checklist técnico ao relatório pronto para entregar

- Metodologia aplicável em cenário real
- Headers, TLS, cookies, CORS e infraestrutura de configuração
- Vulnerabilidades comuns (OWASP) com exemplos e correção
- Checklist prático de auditoria + modelo de relatório profissional
- Inclui a ferramenta CLI companion: reconpp (auditoria automática)

Autor: bryanrbr

Uso autorizado é critério para aplicar qualquer conteúdo. Auditando alvos sem permissão você comete crime. Este guia é focado em defesa e auditoria consentida.

1. Antes de tudo

1.1 Regra de ouro

Auditar um sistema sem autorização escrita do dono é crime em praticamente todos os países, incluindo o Brasil (Lei 12.737/2012 e Marco Civil da Internet). Em um projeto de consultoria:

- Colete um **termo de autorização** ou **escopo assinado** antes de tocar o alvo.
- Documente o escopo (hosts, IPs, domínios) e o intervalo de testes.
- Prefira **repasses por um laboratório homologado** ou ambiente próprio nas primeiras execuções.

1.2 O que este guia entrega

Uma rotina de auditoria de segurança web que **pode ser executada em um único dia**, do primeiro reconhecimento ao relatório final. Ela combina:

- Checagens **passivas e automáticas** (a ferramenta `reconpp` do Apêndice A resolve boa parte em minutos).
- Análise de **configuração** (headers, TLS, cookies, CORS).
- Um **checklist de 70+ pontos** para alvos críticos.
- Um **modelo de relatório** que clientes respeitam.

1.3 Perfil de auditoria

As verificações deste guia são de dois tipos:

Tipo	Exemplos	Risco para o alvo
Passiva	Headers, TLS, cookies, arquivos expostos	Mínimo
Ativa leve	Listagem de diretórios, métodos HTTP, erros	Baixo
Ativa completa	Fuzzing, brute force, exploitation	Média/Alta - requer autorização explícita

Regra prática: **nunca** execute itens de risco médio/alto sem autorização por escrito. A linha do passivo é muito mais ampla do que a maioria imagina - e gera a maior parte dos achados de configuração.

2. Fundamentos: superfície de ataque moderna

2.1 O que um auditor passivo enxerga

Toda resposta HTTP que um navegador recebe contém informação. Antes de qualquer exploração, você já consegue descobrir:

- **Tecnologia do servidor** (cabeçalhos `Server` e `X-Powered-By`).
- **Postura de segurança** (headers de proteção presentes ou ausentes).
- **Gerenciamento de sessão** (atributos dos cookies).

- **Chamadas externas** (CORS configurado).
- **Segurança de transporte** (certificado e protocolos TLS).
- **Arquivos esquecidos** (`.env`, `.git`, backups que respondem 200).

Mapeie isso primeiro. É o "raio-X" inicial de qualquer alvo.

2.2 Headers de proteção: o que importa e o que quebra

Header	Protege contra	Recomendação
---	---	---
Strict-Transport-Security	Downgrade HTTPS→HTTP	max-age=31536000; includeSubDomains; preload
Content-Security-Policy	XSS, data injection	default-src 'self'; script-src 'self'; object-src 'none'; frame-ancestors 'none'
X-Content-Type-Options	MIME sniffing	nosniff
X-Frame-Options	Clickjacking	DENY (ou SAMEORIGIN quando permitido)
Referrer-Policy	Vazamento de referrer	strict-origin-when-cross-origin
Permissions-Policy	APIs via navegador	camera=(), microphone=(), geolocation=()

Armadilha comum: CSP com `unsafe-inline` ou `unsafe-eval` elimina quase todo o ganho de segurança. Se o app precisa deles, o correto é migrar para nonces/hashes, não aceitar a fragilidade.

2.3 TLS: além do "cadeado fechado"

- **Certificado expirado** é reprovado por clientes; renovação deve estar automatizada.
- **TLS 1.0/1.1 aceito** é falha crítica em auditoria de prestígio.
- **Cifras < 128 bits ou RC4/3DES** devem sumir.
- Use `ssl.sh` / `testssl.sh` para verificação completa se precisar de profundidade (a ferramenta companion checa o essencial).

2.4 Cookies: os três atributos que salvam

- `Secure` — só trafega via HTTPS.
- `HttpOnly` — JS não lê (mitiga roubo via XSS).
- `SameSite=Lax|Strict` — mitiga CSRF e side-channel do fetch.
- `SameSite=None` exige `Secure` — e só para casos reais de terceiros.

Cookie de sessão sem `Secure` e sem `HttpOnly` em aplicação financeira é o clássico achado "fácil de certificar, impossível de contestar" — a correção é de configuração, não de código.

3. Vulnerabilidades comuns: detectar e corrigir

3.1 Injection (SQL/NoSQL/LDAP)

Detecção (teste seguro, em autorização):

- Entrada com aspas simples em campo numérico: erro de banco é indicativo (teste manual, com cuidado).
- `ordem de erro`: somente `'` e variantes; nunca usamos `' OR 1=1` + comentários em produção sem autorização formal.

Correção:

- Parametrização obrigatória (prepared statements / ORM).
- Nunca montar string com input do usuário.
- Input validation server-side (tipos, tamanho, charset).

3.2 XSS (Refletido, Armazenado, DOM)

Detecção leve: coloque payloads marcadores (ex.: `qr7t5` tags alternadas) em parâmetros refletidos e observe o HTML retornado - **sem executar nada**. Se o marcador aparece sem escape, o campo é candidato; a evidência é o HTML cru, não um alerta.

Correção:

- Escape na renderização (context-aware, ex.: autoescape de frameworks).
- CSP estrito como camada final.
- Sanitização server-side nunca confiável sozinha.

3.3 CSRF

Detecção passiva:

- Cookie de sessão sem `SameSite`.
- Formulários sem token anti-CSRF nas mutações (POST/DELETE).
- `Origin`/`Referer` não validados.

Correção: tokens anti-CSRF + `SameSite=Lax` + validação de `Origin` em estado de produção.

3.4 SSRF

Detecção: parâmetros de URL/URL de callback (webhooks, importadores, preview links) que aceitam endereços arbitrários. Em teste autorizado, use um host de observação (`canary`) e veja se ele é atingido.

Correção:

- Allowlist de hosts/portas.
- Resolver DNS e **revalidar** o IP antes da requisição (evitar rebinding).
- Bloquear faixas de infraestrutura (169.254.169.254, 10.x, 127.0.0.1).

3.5 IDOR/BOLA

Detecção passiva: trocar `id=1` por `id=2` em recurso, com dois cenários (o auditor, com 2 contas de teste em ambiente autorizado). Falta de checagem de autorização no objeto é o achado.

Correção: autorização por objeto (não por página), testes de acesso por id, e políticas server-side por recurso.

3.6 Upload de arquivo

Detecção: comportamento do servidor com extensões `.svg`, `.html` e dupla extensão — sempre em alvo autorizado.

Correção:

- Desativar execução no diretório de upload.
- Validar MIME + extensão + magic bytes.
- Renomear arquivos com hash; nunca confiar no nome do cliente.
- Storage externo (S3/GCS) quando possível.

3.7 Deserialização e dependências

- Componha um SBOM (Software Bill of Materials) e mapeie versões contra o NVD.
- `npm audit` / `pip-audit` / `trivy` no CI são baratos e dobram como evidência de maturidade.
- Deserialização de dados não confiáveis (pickle, Java serialization, PHP unserialize sem allowed_classes) é grave: evite formatos binários de entrada; use JSON.

3.8 Erros que vazam

- Stack traces públicas (Spring whitelabel, debug pages) revelam versões de libs, caminhos do servidor e query plans.
- **Correção:** logs no servidor, mensagens genéricas para o usuário, suporte para "request id".

3.9 Contas e credenciais

- Testar usuários padrão (`admin/admin`) em instâncias próprias; regra de rotina em consultoria.
- Verificar cadastro em vazamentos (Have I Been Pwned) para identificar credenciais reutilizadas em alvos do cliente — com autorização.
- **Correção:** MFA, senha forte, credencial nos cofres, rotatividade.

3.10 Mapa de severidades

Severidade	Exemplo típico	Impacto
Alta	RCE, SQL Injection, IDOR financeiro	Perda de dados/controle
Média	XSS armazenado, SSRF com limitação, erro de status	Comprometimento parcial
Baixa	HSTS ausente, fingerprint	Superfície ampliada
Info	Server header, robots.txt	Contexto do alvo

4. O checklist prático (70+ pontos)

Módulos na ordem de execução. Marque/pontue para o alvo. Use a ferramenta companion para os blocos A-G de uma vez.

4.1 A - Configuração de transporte

- [ ] Site responde somente via HTTPS
- [ ] HSTS com max-age>= 180 dias, includeSubDomains
- [ ] Sem downgrade HTTPS→HTTP na cadeia de redirecionamento
- [ ] Certificado válido, pelo menos 14 dias de vida
- [ ] TLS 1.2/1.3 apenas (1.0/1.1 desativados)
- [ ] Sem cifras < 128 bits (RC4/3DES/CBC vulnerável)
- [ ] Preload list considerado (hstspreload.org)

4.2 B - Headers de proteção

- [ ] CSP com resultados e migração p/ nonces (sem unsafe-inline/eval)
- [ ] X-Content-Type-Options: nosniff
- [ ] X-Frame-Options: DENY/SAMEORIGIN ou frame-ancestors
- [ ] Referrer-Policy: strict-origin-when-cross-origin
- [ ] Permissions-Policy limitado
- [ ] Sem X-Powered-By/Servidor identificável
- [ ] cors: sem reflexo arbitrário, sem wildcard+credentials

4.3 C - Sessions

- [ ] Cookies com Secure, HttpOnly, SameSite (Lax/Strict)
- [ ] Session ID: novo login = novo ID
- [ ] Tempo de login com idle timeout e expiração global
- [ ] Logout efetivo no servidor
- [ ] Proteção contra fixation (regenerar sessão)

4.4 D - API e dados

- [ ] Rate limiting em endpoints sensíveis (login, OTP, upload)
- [ ] Paginação/limites em listagens (evitar data leaks massivos)
- [ ] Erros genéricos; sem stack traces públicas
- [ ] CORS restrito a domínios confiáveis
- [ ] Payload máximo e validação por tipo
- [ ] Logs de segurança: falhas de auth, anomalias, acessos admin
- [ ] Secrets fora do código (env/SDK/cofres); invalidação rotineira

4.5 E - Superfície exposta

- [ ] Nenhum `.env`, `.git`, `dump.sql`, `backup.zip`, `phpinfo.php`, `server-status` (HTTP 200)
- [ ] `security.txt` (RFC 9116) publicado
- [ ] Nenhum painel administrativo exposto (login antes)

- [ ] Diretórios bloqueados ao acesso não autorizado
- [ ] Cabeçalhos de cache corretos

4.6 F - Auth & OAuth

- [ ] MFA habilitado e não contornável (downgrade de fluxo)
- [ ] Debugger de SMS/email limitado — sem contorno por atalho
- [ ] Redirect URIs em allowlist no fluxo OAuth
- [ ] JWTs com algortimos fixados, sem `alg: none`, exp curta
- [ ] Password reset com token único, tempo curto, uso único

4.7 G - Dependências e deploy

- [ ] SBOM dentro do CI
- [ ] Varredura (trivy/npm audit/pip-audit) em todo PR
- [ ] Secret scanning (gitleaks/guardrails) no push
- [ ] Imagens de produção sem shell/ferramentas desnecessárias
- [ ] Patches de segurança aplicados em menos de 72h (criticidade alta)

4.8 H - Humanos & continuidade

- [ ] Web-fim: política de senha + treinamento de phishing contínuo
- [ ] Incidente: playbook com responsáveis e tempos
- [ ] Auditoria periódica: frequência anual (ou 90 dias após mudanças)

5. O relatório que o cliente respeita

5.1 Estrutura vencedora

1. **Escopo:** URLs, hosts, período, objetivo.
2. **Metodologia:** o que foi feito, e o que NÃO foi feito (limitações explícitas evitam disputa de expectativa).
3. **Resumo executivo:** 5-8 linhas para o diretor decidir. Achados total + % alta/média.
4. **Tabela de achados:** ID, severidade, nome, afeta, impacto, risco.
5. **Detalhamento:** por achado — descrição, evidência, passo-a-passo de reprodução, correção recomendada, alternativa.
6. **Anexos:** requisições, saída de ferramenta, checklist pontuado.

5.2 Regras de evidência

- Nunca altere dados de produção para provar algo (ex.: injete no registro de um terceiro). Use dados de teste criados por você.
- Guarde request/response completos (curl -i ou mitm) — sem pressa para remover.

- Severidade consolidada com contexto do negócio: um IDOR de dados financeiro é maior risco num produto B2B que num blog.
- O time de correção precisa de reprodução: dê *exact steps*, não só screenshots.

5.3 Exemplo de entrada de achado

ID-01 | Médio | Cookie de sessão sem Secure e HttpOnly

- **Componente:** https://cliente.com/login
- **Descrição:** o cookie JSESSIONID é emitido sem atributos. Em rede não confiável, pode ser captiveado (Secure) — e é lido por qualquer XSS (HttpOnly).
- **Evidência (curl -i):**
- **Correção:** configurar cookie flag no framework (ex.: `server.servlet.session.cookie`); verificar reversão.
- **Bibliografia:** OWASP Session Management Cheat Sheet.

5.4 Como debriefar com o cliente

- Agende reunião de 30 min — 5 min de achados grandes, 10 de detalhes, 15 de priorização.
- Priorize a pasta "quick wins" de configuração (headers/cookies/TLS): a maioria se resolve em horas.
- Coloque-se como parte da solução: proponha retest.
- NÃO minimize nem maximize: contexto é tudo.

Apêndice A - Ferramenta companion: reconpp

`reconpp` é um CLI em Python (stdlib puro, zero dependências) que executa as checagens dos blocos A-E em um minuto e produz relatórios em markdown/JSON prontos para entregar.

Checagens: HSTS (idade/subdomains), CSP (unsafe-inline/eval, default-src), X-Content-Type-Options, X-Frame-Options, clickjacking, Referrer-Policy, Permissions-Policy, fingerprinting Server/X-Powered-By, downgrade HTTPS→HTTP, cookies (Secure/HttpOnly/SameSite), CORS (origem refletida, wildcard, credentials), TLS (expiração, TLS 1.0/1.1, cifra <128 bits), expostos (.env, .git, backups, dumps, actuator), well-known (robots, security.txt RFC 9116, sitemap).

```
pip install .
reconpp -u https://cliente.com -f md -o report.md
reconpp -u cliente.com -f json
```

Tudo **passivo** — sem fuzzing, sem brute force, sem exploitation.

Apêndice B - Sanidade e limites operacionais

- Use sempre redes separadas (scan máquina VM/container) para não sujar seu ambiente de trabalho.
- Limite a vazão (rate), programas sempre com atrasos para não derrubar o alvo.
- Combine horários com o cliente; nunca impacte produção próxima a janelas críticas.

- Grave a "linha do tempo" da auditoria: minutos, comandos, evidências.

Apêndice C - Fontes oficiais para profundidade

- OWASP Top 10 - owasp.org
- OWASP WSTG (test guide completo interativo)
- OWASP Cheat Sheet Series (headers, sessions, CSRF, XSS)
- CWE (MITRE): classificação e verificação
- CVE/NVD: referência de vulnerabilidades conhecidas
- ferramentas: testssl.sh, nuclei (template-based, escopo autorizado), trivy, gitleaks

Este guia não substitui auditoria profissional. Use responsabilidade: autorização por escrito sempre.