ClawGuru

Hardening Sprint Pack

30–60 Min. Security Hardening

Copy/Paste Templates · Checklists · Baselines

Version 1.0 · February 2026 · clawguru.org

SSH Hardening	**Firewall Baseline**	**Docker Secrets**
Nginx Hardening	**Security Headers**	**Incident Checklist**

■ **Stop the bleeding first:** Wenn Production betroffen ist: zuerst Containment, dann Root Cause Analysis.

1 · SSH Hardening

SSH ist der häufigste Angriffsvektor auf Linux-Server. Mit diesen Maßnahmen eliminierst du 90% der Angriffsfläche.

- Key-only: Passwort-Login deaktivieren
- Root-Login verbieten – Admin via sudo
- Rate Limiting: MaxAuthTries + fail2ban
- SSH nur via VPN / Tailscale (kein Public 22)
- AllowUsers explizit setzen

sshd_config (Minimal Secure)

```
# /etc/ssh/sshd_config
PermitRootLogin no
PasswordAuthentication no
PubkeyAuthentication yes
MaxAuthTries 3
LoginGraceTime 30
AllowUsers deploy admin
ClientAliveInterval 300
ClientAliveCountMax 2
# Nach Änderung: systemctl reload sshd
```

✓ **Verify:** ssh root@server → muss mit 'Permission denied' scheitern. Eigener Key: muss funktionieren.

fail2ban (SSH Brute Force)

```
# /etc/fail2ban/jail.local
[sshd]
enabled = true
maxretry = 3
bantime  = 3600
findtime = 600
# systemctl enable --now fail2ban && fail2ban-client status sshd
```

2 · Firewall Baseline (UFW)

Default deny inbound. Nur was explizit erlaubt ist, darf rein. SSH extern nur über VPN oder Allowlist.

```
# UFW Baseline
ufw default deny incoming
ufw default allow outgoing
ufw allow 80/tcp    # HTTP
ufw allow 443/tcp   # HTTPS
# SSH nur von eigener IP:
# ufw allow from 1.2.3.4 to any port 22 proto tcp
ufw enable
ufw status verbose
```

■ **DB-Ports:** Niemals Port 5432 (Postgres), 6379 (Redis), 27017 (Mongo) oeffentlich oeffnen.

Docker + UFW Konflikt

```
# Docker umgeht UFW – daher in daemon.json:
{ "iptables": false }
# Dann UFW-Regeln fuer Docker-Netz manuell setzen
# Oder: github.com/chaifeng/ufw-docker verwenden
```

3 · Nginx Hardening

Nginx als Reverse Proxy: TLS erzwingen, Server-Info verstecken, Rate Limits, Security Headers.

nginx.conf – Sichere Defaults

```
# http block
server_tokens off;
client_max_body_size 10m;
client_body_timeout 12;
client_header_timeout 12;
keepalive_timeout 15;
limit_req_zone $binary_remote_addr zone=general:10m rate=10r/s;
limit_req_zone $binary_remote_addr zone=login:10m rate=5r/m;
```

server block – TLS + Headers

```
server {
  listen 443 ssl http2;
  ssl_protocols TLSv1.2 TLSv1.3;
  ssl_prefer_server_ciphers off;

  add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;
  add_header X-Frame-Options "DENY" always;
  add_header X-Content-Type-Options "nosniff" always;
  add_header Referrer-Policy "no-referrer-when-downgrade" always;

  location /login {
    limit_req zone=login burst=5 nodelay;
    proxy_pass http://localhost:3000;
  }
}
```

✓ **Test:** curl -I https://deine-domain.de | grep -E 'Strict|Frame|Content-Type'

4 · Docker Secrets & ENV Hygiene

- Nie .env in Git commiten (.gitignore pruefen!)
- docker-compose: Secrets als files

- Next.js: NEXT_PUBLIC_* nur fuer echte oeffentliche Werte
- CI/CD: Secrets im Secret Store (GitHub, Netlify, Vercel)
- Rotation Plan: Keys regelmaessig rotieren

```
# docker-compose.yml – Secrets richtig
services:
  app:
    image: myapp:latest
    secrets: [db_password]
    # Kein DB_PASSWORD als ENV!
secrets:
  db_password:
    file: ./secrets/db_password.txt  # Nie in Git!
```

```
# .gitignore – Mindest-Eintraege
.env
.env.*
!.env.example
secrets/
*.pem
*.key
*_rsa
```

---

5 · Security Sprint Checklist

Arbeite diese Liste in 30-60 Minuten durch. Jedes Haekchen reduziert Angriffsflaeche.

Status	Check	Prio
[ ]	SSH: Key-only, kein Root-Login, fail2ban aktiv	KRITISCH
[ ]	Firewall: Default deny, DB-Ports geschlossen	KRITISCH
[ ]	TLS: HTTPS erzwungen, kein HTTP	HOCH
[ ]	Security Headers: HSTS, X-Frame, nosniff	HOCH
[ ]	Nginx: server_tokens off, Rate Limits gesetzt	HOCH
[ ]	Docker: Keine Secrets als ENV vars	HOCH
[ ]	.gitignore: .env, keys, secrets excluded	KRITISCH
[ ]	Updates: OS + Docker + App aktuell	MITTEL
[ ]	Backup: Taeglich + Restore getestet	HOCH
[ ]	API Keys: Rotiert, Scope minimal	HOCH
[ ]	DB: Least Privilege, kein Public Access	KRITISCH

---

6 · Stripe Webhook Verification

Stripe-Webhooks ohne Signaturpruefung sind ein kritisches Sicherheitsloch. Immer verifizieren.

```
// /app/api/stripe/webhook/route.ts
export const runtime = "nodejs"
export async function POST(req: Request) {
  const sig = req.headers.get("stripe-signature")
  const body = await req.arrayBuffer()
  let event: Stripe.Event
  try {
```

```
    event = stripe.webhooks.constructEvent(
      Buffer.from(body), sig!, process.env.STRIPE_WEBHOOK_SECRET!
    )
  } catch { return new Response("Invalid signature", { status: 400 }) }
  // idempotent: store event.id, process once
  return new Response("ok")
}
```

✓ **Stripe CLI:** stripe listen --forward-to localhost:3000/api/stripe/webhook

7 · API Key Rotation (Notfall)

Key geleakt? Hier ist der Ablauf. Schnelligkeit ist entscheidend.

Schritt	Aktion	Zeit
1	Alten Key sofort in Stripe/Service deaktivieren	< 5 Min
2	Neuen Key generieren (Scope minimal!)	< 2 Min
3	Neuen Key in allen Envs setzen (Prod + CI/CD)	< 10 Min
4	Rollout: Deploy, Health-Check, Smoke Test	< 15 Min
5	Audit: Was hat der alte Key zugegriffen?	30 Min
6	Post-Mortem: Wie ist Key in Git/Log gelangt?	1-2h

```
# Pruefen ob Key im Git-Verlauf liegt:
git log --all -S "sk_live_" --oneline | head -20
git log --all -S "AKIA" --oneline | head -20
# Oder: gitleaks detect --source . --verbose
```

Next Steps

- Security Score pruefen: clawguru.org/check
- Copilot: Konkretes Runbook fuer dein Problem: /copilot
- Vault: Alle Templates + Runbooks: /vault
- Incident Kit Pro (ZIP): Key-Rotation, WAF, Cloudflare: /downloads
- Badge teilen: /badge