16.6 Schwachstellen, Patchmanagement und Hardening

Schwachstellen sind Sicherheitslücken, die von Angreifern ausgenutzt werden können.

Sie können entstehen durch:

- Programmierfehler
- veraltete Software
- unsichere Standardkonfiguration
- schwache Passwörter
- fehlende Zugriffskontrolle
- offene Ports
- unnötige Dienste
- falsche Berechtigungen
- unsichere Protokolle
- fehlende Updates
- menschliche Fehler

Merksatz:

> Eine Schwachstelle ist eine ausnutzbare Sicherheitslücke.

Warum Schwachstellen gefährlich sind

Schwachstellen können dazu führen, dass Angreifer:

- Systeme übernehmen
- Daten auslesen
- Daten verändern
- Dienste stören
- Malware einschleusen
- Rechte ausweiten
- interne Systeme erreichen
- Sicherheitsfunktionen umgehen

Merksatz:

Schwachstellen sind oft der Einstiegspunkt für Angriffe.

Schwachstelle, Bedrohung und Risiko unterscheiden

Begriff	Bedeutung	Beispiel
Schwachstelle	ausnutzbare Lücke	ungepatchter Server
Bedrohung	mögliches schädliches Ereignis	Angreifer nutzt Lücke aus
Risiko	Wahrscheinlichkeit und Auswirkung	Serverübernahme mit Datenabfluss

Merksatz:

Schwachstelle ist die Lücke.
Bedrohung ist das mögliche Ereignis.
Risiko ist die bewertete Gefahr.

Beispiele für technische Schwachstellen

Typische technische Schwachstellen:

veraltetes Betriebssystem
ungepatchter Webserver
veraltete VPN-Appliance
unsichere TLS-Version
Standardpasswort
offener RDP-Port
öffentlich erreichbare Datenbank
unsichere API
fehlende Eingabeprüfung
fehlerhafte Zugriffskontrolle
veraltete Bibliothek
unsicheres Container-Image

Merksatz:

Technische Schwachstellen entstehen oft durch alte Versionen oder falsche Konfiguration.

Beispiele für organisatorische Schwachstellen

Organisatorische Schwachstellen sind zum Beispiel:

```
keine Zuständigkeiten
keine regelmäßigen Updates
keine Rechteprüfung
kein Backup-Konzept
keine Restore-Tests
keine Sicherheitsrichtlinie
kein Offboarding-Prozess
keine Dokumentation
keine Notfallplanung
keine Schulung
```

Merksatz:

```
Organisation kann genauso Schwachstelle sein wie Technik.
```

Beispiele für menschliche Schwachstellen

Menschliche Schwachstellen entstehen durch:

```
fehlendes Sicherheitsbewusstsein
Passwort-Wiederverwendung
unkritisches Öffnen von Anhängen
Bestätigen unbekannter MFA-Anfragen
Weitergabe von Zugangsdaten
Umgehen von Prozessen
fehlende Schulung
Zeitdruck
Gewohnheit
```

Merksatz:

```
Menschen brauchen klare Prozesse und Schulung,
nicht nur technische Regeln.
```

CVE

CVE steht für:

```
Common Vulnerabilities and Exposures
```

CVE ist eine standardisierte Kennung für bekannte Schwachstellen.

Beispielprinzip:

```
CVE-Jahr-Nummer
```

Eine CVE beschreibt eine bekannte Schwachstelle, damit Hersteller, Administratoren und Sicherheitswerkzeuge eindeutig darüber sprechen können.

Merksatz:

```
CVE ist eine eindeutige Kennung für bekannte Schwachstellen.
```

CVSS

CVSS steht für:

```
Common Vulnerability Scoring System
```

CVSS bewertet, wie kritisch eine Schwachstelle ist.

Der Wert liegt üblicherweise zwischen:

```
0,0
und
10,0
```

Je höher der Wert, desto kritischer ist die Schwachstelle.

Merksatz:

```
CVSS bewertet die Kritikalität einer Schwachstelle.
```

CVSS richtig einordnen

Ein hoher CVSS-Wert bedeutet:

```
Schwachstelle ist technisch kritisch.
```

Aber für die Praxis muss zusätzlich geprüft werden:

```
Ist das System betroffen?
Ist der Dienst erreichbar?
Gibt es bereits Ausnutzung?
Gibt es Schutzmaßnahmen?
Sind sensible Daten betroffen?
Ist das System geschäftskritisch?
Gibt es einen Patch?
Gibt es einen Workaround?
```

Merksatz:

```
CVSS hilft bei Priorisierung,
ersetzt aber keine Risikoanalyse.
```

Exploit

Ein Exploit ist eine Methode oder ein Code, mit dem eine Schwachstelle ausgenutzt wird.

Beispiele:

```
Exploit für veralteten Webserver
Exploit für ungepatchte VPN-Appliance
Exploit für fehlerhafte Bibliothek
Exploit für unsichere Eingabeprüfung
```

Merksatz:

```
Exploit = Ausnutzung einer Schwachstelle.
```

Zero-Day-Schwachstelle

Eine Zero-Day-Schwachstelle ist eine Schwachstelle, für die noch kein Patch verfügbar ist oder die dem Hersteller noch nicht bekannt ist.

Ein Zero-Day-Angriff nutzt diese Schwachstelle aus.

Schutz ist schwieriger, aber möglich durch:

```
Defense in Depth
Least Privilege
Netzwerksegmentierung
EDR
WAF
Monitoring
Härtung
schnelle Reaktion
```

Merksatz:

```
Zero-Day bedeutet:
Angriff vor verfügbarem regulärem Patch.
```

Patch

Ein Patch ist eine Korrektur oder Aktualisierung.

Ein Patch kann beheben:

```
Sicherheitslücke
Programmfehler
Stabilitätsproblem
Kompatibilitätsproblem
```

Merksatz:

```
Patch = Korrektur oder Sicherheitsupdate.
```

Update und Upgrade unterscheiden

Update:

kleinere Aktualisierung
Fehlerbehebung
Sicherheitskorrektur
Versionspflege

Upgrade:

größere Versionsänderung
neue Hauptversion
neue Funktionen
größere technische Änderung

Merksatz:

Update ist meist kleiner.
Upgrade ist meist größer.

Patchmanagement

Patchmanagement bedeutet:

Updates werden geplant,
geprüft,
verteilt
und kontrolliert.

Patchmanagement betrifft:

Betriebssysteme
Anwendungen
Browser
Serverdienste
Datenbanken
Firewalls
Router
Switches
VPN-Gateways

Firmware
Container-Images
Bibliotheken
Cloud-Systeme

Merksatz:

Patchmanagement ist die organisierte Verwaltung von Sicherheitsupdates.

Warum Patchmanagement wichtig ist

Viele erfolgreiche Angriffe nutzen bekannte Schwachstellen, für die bereits Updates existieren.

Ohne Patchmanagement bleiben Systeme unnötig angreifbar.

Typische Folgen:

Malware-Infektion
Ransomware
Datenabfluss
Systemübernahme
Ausfall
laterale Bewegung im Netzwerk

Merksatz:

Ungepatchte Systeme sind leichte Ziele.

Patchmanagement-Ablauf

Ein sinnvoller Ablauf:

1. Systeme inventarisieren.
2. Schwachstelleninformationen prüfen.
3. Betroffenheit bewerten.
4. Kritikalität priorisieren.
5. Patch testen.
6. Wartungsfenster planen.

7. Patch ausrollen.
8. Funktion prüfen.
9. Erfolg dokumentieren.
10. Rest-Risiken bewerten.

Merksatz:

Patchmanagement braucht Überblick,
Priorisierung,
Test
und Dokumentation.

Inventarisierung

Inventarisierung bedeutet:

Es wird erfasst,
welche Systeme,
Software,
Versionen
und Komponenten vorhanden sind.

Ohne Inventar ist unklar:

Welche Systeme sind betroffen?
Welche Version läuft?
Wer ist verantwortlich?
Wo muss gepatcht werden?
Welche Systeme sind kritisch?

Merksatz:

Was nicht bekannt ist,
kann nicht zuverlässig gepatcht werden.

Asset

Ein Asset ist ein Wert oder eine Ressource, die geschützt werden muss.

Beispiele:

```
Server
Notebook
Anwendung
Datenbank
Firewall
Switch
Cloud-Ressource
Benutzerkonto
Datenbestand
Zertifikat
API-Schlüssel
```

Merksatz:

```
Asset = schützenswerte Ressource.
```

Asset Management

Asset Management bedeutet:

```
IT-Ressourcen werden erfasst,
verwaltet
bewertet
und über ihren Lebenszyklus verfolgt.
```

Wichtig für:

```
Patchmanagement
Lizenzmanagement
Sicherheitsbewertung
Kostenkontrolle
Verantwortlichkeiten
Notfallplanung
```

Merksatz:

Asset Management ist Grundlage für geordneten IT-Betrieb.

Priorisierung von Patches

Nicht alle Patches können immer gleichzeitig installiert werden.

Priorisierung nach:

Kritikalität der Schwachstelle
Ausnutzbarkeit
betroffene Systeme
öffentliche Erreichbarkeit
Geschäftskritikalität
vorhandene Schutzmaßnahmen
bekannte Angriffe
Datenklassifizierung
Aufwand und Risiko des Updates

Merksatz:

Kritische,
ausnutzbare
und öffentlich erreichbare Systeme zuerst patchen.

Notfallpatch

Ein Notfallpatch wird schnell eingespielt, wenn eine Schwachstelle besonders kritisch ist.

Beispiele:

aktive Ausnutzung bekannt
öffentlich erreichbarer Dienst betroffen
Ransomware nutzt Lücke aus
keine ausreichende Schutzmaßnahme vorhanden

Merksatz:

Kritische Schwachstellen können außerplanmäßige Updates erfordern.

Wartungsfenster

Ein Wartungsfenster ist ein geplanter Zeitraum, in dem Änderungen durchgeführt werden.

Ziele:

```
Benutzer informieren
Ausfallzeit planen
Risiken reduzieren
Rollback vorbereiten
Verantwortliche erreichbar halten
```

Merksatz:

```
Wartungsfenster machen Änderungen planbar.
```

Rollback

Rollback bedeutet:

```
Eine Änderung wird zurückgenommen.
```

Beispiele:

```
Patch deinstallieren
Snapshot zurückspielen
Konfiguration wiederherstellen
alte Version aktivieren
Backup wiederherstellen
```

Wichtig:

```
Rollback muss vor der Änderung geplant sein.
```

Merksatz:

```
Jede kritische Änderung braucht einen Rückweg.
```

Patch testen

Patches sollten geprüft werden, besonders bei kritischen Systemen.

Zu prüfen:

```
startet das System?
läuft die Anwendung?
funktionieren Dienste?
gibt es Kompatibilitätsprobleme?
funktionieren Schnittstellen?
sind Performance-Probleme entstanden?
funktioniert Backup weiterhin?
```

Merksatz:

```
Patchen ohne Test kann neue Störungen erzeugen.
```

Patch-Erfolg prüfen

Nach dem Patchen muss geprüft werden:

```
Version aktualisiert?
Schwachstelle geschlossen?
Dienste laufen?
Logs unauffällig?
Monitoring grün?
Benutzer können arbeiten?
Schwachstellenscan bestätigt Erfolg?
```

Merksatz:

```
Patch abgeschlossen heißt:
Wirkung und Betrieb geprüft.
```

Patchmanagement und Dokumentation

Dokumentiert werden sollten:

```
betroffene Systeme
alte Version
neue Version
Zeitpunkt
Verantwortlicher
Grund
Testergebnis
Rollback-Plan
bekannte Probleme
offene Rest-Risiken
```

Merksatz:

```
Dokumentation macht Änderungen nachvollziehbar.
```

Firmware-Updates

Firmware ist Software, die direkt auf Geräten läuft.

Betroffene Geräte:

```
Router
Switches
Firewalls
Access Points
Drucker
NAS
Serverhardware
SSDs
IoT-Geräte
```

Firmware-Updates sind wichtig, weil Netzwerkgeräte oft dauerhaft erreichbar und sicherheitskritisch sind.

Merksatz:

```
Auch Netzwerkgeräte und Firmware müssen gepatcht werden.
```

Patchmanagement bei Netzwerkgeräten

Bei Netzwerkgeräten besonders beachten:

- Konfiguration sichern
- Firmware-Kompatibilität prüfen
- Wartungsfenster planen
- Redundanz prüfen
- Zugriff nach Update testen
- Rollback-Möglichkeit prüfen
- Monitoring beobachten

Merksatz:

> Netzwerkgeräte-Updates brauchen besondere Vorsicht,
> weil sie Verbindungen beeinflussen.

Patchmanagement bei Servern

Bei Servern beachten:

- Dienste prüfen
- Abhängigkeiten prüfen
- Backups erstellen
- Wartungsfenster planen
- Neustarts berücksichtigen
- Anwendungskompatibilität prüfen
- Monitoring nach Update prüfen

Merksatz:

> Serverpatches betreffen oft mehrere Dienste gleichzeitig.

Patchmanagement bei Clients

Bei Clients beachten:

Betriebssystemupdates
Browserupdates
Office-Updates
PDF-Reader
E-Mail-Client
VPN-Client
Sicherheitssoftware
Treiber
lokale Anwendungen

Merksatz:

Clients sind häufige Einstiegspunkte und müssen aktuell bleiben.

Patchmanagement bei Cloud

Cloud-Patchverantwortung hängt vom Service-Modell ab.

IaaS:

Kunde patcht Betriebssystem,
Anwendungen
und eigene Dienste.

PaaS:

Anbieter patcht Plattform,
Kunde patcht Anwendung,
Code
und Abhängigkeiten.

SaaS:

Anbieter patcht Anwendung,
Kunde prüft Einstellungen,
Benutzer,
Rechte
und Sicherheit.

Merksatz:

> Cloud nimmt nicht jede Patchverantwortung ab.

Patchmanagement bei Containern

Container müssen ebenfalls gepflegt werden.

Zu prüfen:

> Basisimage
> Bibliotheken
> Laufzeitumgebung
> Anwendungscode
> Paketabhängigkeiten
> Secrets im Image
> unsichere Konfiguration
> Container-Registry

Maßnahmen:

> Images regelmäßig neu bauen
> Images scannen
> minimale Images nutzen
> veraltete Images ersetzen
> keine unnötigen Pakete installieren

Merksatz:

> Container-Images altern und müssen aktualisiert werden.

Dependency Management

Dependency Management bedeutet:

> Abhängigkeiten einer Software werden verwaltet.

Beispiele:

Bibliotheken
Frameworks
Pakete
Module
Container-Basisimages
Plugins

Risiko:

Eine verwundbare Bibliothek kann die ganze Anwendung gefährden.

Merksatz:

Auch Abhängigkeiten sind Teil der Sicherheit.

SBOM

SBOM steht für:

Software Bill of Materials

Eine SBOM listet Softwarebestandteile einer Anwendung auf.

Beispiele:

Bibliotheken
Versionen
Komponenten
Abhängigkeiten

Nutzen:

schneller erkennen,
ob eine bekannte Schwachstelle betroffen ist.

Merksatz:

SBOM ist eine Stückliste für Softwarebestandteile.

Schwachstellenscan

Ein Schwachstellenscan sucht automatisch nach bekannten Schwachstellen.

Er prüft zum Beispiel:

offene Ports
Dienstversionen
bekannte CVEs
fehlende Patches
unsichere TLS-Konfiguration
Standarddienste
schwache Konfigurationen
verwundbare Bibliotheken

Merksatz:

Schwachstellenscan findet bekannte technische Probleme.

Grenzen von Schwachstellenscans

Ein Schwachstellenscan erkennt nicht alles.

Grenzen:

unbekannte Schwachstellen
Logikfehler in Anwendungen
komplexe Rechtefehler
falsch bewertete Ergebnisse
fehlende Zugangsdaten beim Scan
Fehlalarme
nicht erreichbare Systeme

Merksatz:

Schwachstellenscan unterstützt,
ersetzt aber keine Sicherheitsanalyse.

False Positive und False Negative

False Positive:

Ein Problem wird gemeldet,
obwohl es nicht wirklich besteht.

False Negative:

Ein Problem wird nicht gemeldet,
obwohl es besteht.

Merksatz:

Scan-Ergebnisse müssen bewertet werden.

Penetrationstest

Ein Penetrationstest ist eine kontrollierte Sicherheitsprüfung, bei der Angriffe simuliert werden.

Ziel:

Schwachstellen realistisch prüfen
Angriffspfade erkennen
Auswirkungen bewerten
Maßnahmen ableiten

Wichtig:

schriftlicher Auftrag
definierter Umfang
erlaubter Zeitraum
klare Regeln
Abschlussbericht

Merksatz:

> Penetrationstest ist kontrollierte Angriffssimulation.

Schwachstellenscan und Penetrationstest vergleichen

Merkmal	Schwachstellenscan	Penetrationstest
Art	automatisiert	manuell und automatisiert
Ziel	bekannte Schwachstellen finden	Angriffswege bewerten
Tiefe	eher breit	tiefer und kontextbezogener
Ergebnis	Liste möglicher Schwachstellen	bewerteter Sicherheitsbericht
Grenze	Fehlalarme möglich	zeitlich und fachlich begrenzt

Merksatz:

> Scan findet Hinweise.
> Penetrationstest bewertet Angriffsrealität.

Hardening

Hardening bedeutet:

> Systeme werden sicherer konfiguriert,
> indem unnötige Funktionen entfernt
> und sichere Einstellungen gesetzt werden.

Ziel:

> Angriffsfläche reduzieren

Merksatz:

> Hardening reduziert unnötige Angriffsmöglichkeiten.

Warum Hardening wichtig ist

Viele Systeme werden mit Standardfunktionen ausgeliefert, die nicht immer benötigt werden.

Beispiele:

```
Standardkonten
Beispieldateien
Testseiten
offene Dienste
unnötige Module
alte Protokolle
unsichere Standardwerte
Verzeichnislisting
```

Merksatz:

```
Standardkonfiguration ist nicht automatisch sicher.
```

Hardening-Grundregeln

Wichtige Grundregeln:

```
unnötige Dienste deaktivieren
offene Ports reduzieren
Standardpasswörter ändern
Standardkonten deaktivieren
sichere Protokolle nutzen
alte Protokolle abschalten
Rechte minimieren
Logging aktivieren
sichere Konfiguration dokumentieren
Zugriff beschränken
Updates einspielen
```

Merksatz:

```
Nur aktivieren,
was wirklich benötigt wird.
```

Betriebssystem-Hardening

Maßnahmen:

- unnötige Dienste deaktivieren
- lokale Firewall aktivieren
- Benutzerrechte begrenzen
- sichere Passwortrichtlinien
- automatische Updates planen
- Protokollierung aktivieren
- unsichere Freigaben entfernen
- Adminzugriffe begrenzen
- Standardkonten prüfen
- Systemdateien schützen

Merksatz:

> Betriebssysteme brauchen sichere Grundkonfiguration.

Server-Hardening

Maßnahmen:

- nur benötigte Rollen installieren
- Dienste auf notwendige Ports begrenzen
- Adminzugänge beschränken
- TLS korrekt konfigurieren
- Sicherheitsheader setzen
- Fehlerausgaben reduzieren
- Logs aktivieren
- Dateirechte prüfen
- Backup-Zugriff schützen
- Monitoring einrichten

Merksatz:

> Server sollen nur notwendige Dienste bereitstellen.

Webserver-Hardening

Maßnahmen:

- Verzeichnislisting deaktivieren
- unnötige Module entfernen
- alte TLS-Versionen deaktivieren
- Security Headers setzen
- Fehlermeldungen begrenzen
- Standardseiten entfernen
- Upload-Verzeichnisse absichern
- Dateirechte begrenzen
- Adminoberflächen schützen
- Logs aktivieren

Merksatz:

> Webserver nicht mit unsicheren Standardfunktionen betreiben.

Datenbank-Hardening

Maßnahmen:

- nicht öffentlich erreichbar machen
- starke Authentifizierung
- Datenbankbenutzerrechte begrenzen
- Standardkonten deaktivieren
- Verschlüsselung prüfen
- Backups schützen
- Logging aktivieren
- Netzwerkzugriff einschränken
- Sicherheitsupdates einspielen
- keine Testdaten in Produktion

Merksatz:

> Datenbanken gehören besonders geschützt.

Netzwerkgeräte-Hardening

Maßnahmen:

Standardpasswörter ändern
Managementzugang beschränken
unsichere Protokolle deaktivieren
SSH statt Telnet
HTTPS statt HTTP
SNMP sicher konfigurieren
Firmware aktuell halten
ungenutzte Ports deaktivieren
Konfiguration sichern
Logging an zentralen Server senden

Merksatz:

Netzwerkgeräte sind sicherheitskritische Infrastruktur.

Client-Hardening

Maßnahmen:

keine lokalen Adminrechte
automatische Updates
EDR oder Antivirus
Makros einschränken
Application Control
Festplattenverschlüsselung
Bildschirmsperre
sichere Browserkonfiguration
USB-Regeln
Firewall aktivieren

Merksatz:

Clients sind häufige Einstiegspunkte und müssen gehärtet werden.

Cloud-Hardening

Maßnahmen:

MFA erzwingen
IAM minimal berechtigen
öffentliche IPs begrenzen
Security Groups eng setzen
Storage nicht öffentlich freigeben
Audit-Logs aktivieren
Verschlüsselung aktivieren
Schlüsselverwaltung klären
Kostenalarme setzen
ungenutzte Ressourcen löschen
Standardrollen prüfen

Merksatz:

Cloud-Hardening bedeutet sichere Konfiguration von Identitäten,
Netzwerken,
Daten
und Diensten.

Container-Hardening

Maßnahmen:

minimale Images verwenden
keine Root-Ausführung,
wenn nicht nötig
keine Secrets im Image
Images scannen
nur vertrauenswürdige Registries
Schreibrechte begrenzen
Netzwerkzugriffe begrenzen
Ressourcenlimits setzen
unnötige Tools entfernen
Images regelmäßig neu bauen

Merksatz:

```
Container sollten klein,
aktuell
und rechtearm sein.
```

Konfigurationsbaseline

Eine Baseline ist eine festgelegte Standardkonfiguration.

Sie beschreibt, wie ein System sicher eingerichtet sein soll.

Beispiele:

```
erlaubte Dienste
Passwortregeln
Logging-Einstellungen
Firewall-Grundregeln
Update-Einstellungen
TLS-Vorgaben
Rechtevorgaben
```

Merksatz:

```
Baseline = definierter sicherer Grundzustand.
```

Configuration Drift

Configuration Drift bedeutet:

```
Die tatsächliche Konfiguration weicht von der geplanten Baseline ab.
```

Ursachen:

```
manuelle Änderungen
Notfalländerungen
fehlende Dokumentation
unterschiedliche Installationen
```

ungeprüfte Anpassungen

Merksatz:

Configuration Drift macht Systeme uneinheitlich und schwerer sicher zu betreiben.

Compliance-Scan

Ein Compliance-Scan prüft, ob Systeme bestimmte Vorgaben erfüllen.

Beispiele:

Passwortregeln aktiv?
Logging aktiv?
Firewall aktiv?
Verschlüsselung aktiv?
unsichere Dienste deaktiviert?
Standardkonten deaktiviert?
Patchstand korrekt?

Merksatz:

Compliance-Scan prüft Einhaltung definierter Sicherheitsvorgaben.

Change Management bei Sicherheit

Sicherheitsänderungen sollten geplant und dokumentiert werden.

Beispiele:

Firewall-Regel ändern
Adminrolle vergeben
Dienst öffnen
Patch einspielen
Zertifikat erneuern
TLS-Version ändern
neue VPN-Gruppe erstellen

Merksatz:

```
Auch Sicherheitsänderungen brauchen Kontrolle.
```

Vier-Augen-Prinzip

Vier-Augen-Prinzip bedeutet:

```
kritische Aktionen werden von mindestens zwei Personen geprüft oder freigegeben.
```

Beispiele:

```
Adminrechte vergeben
Firewall nach außen öffnen
Backup löschen
Produktivsystem ändern
Zahlungsdaten ändern
kritische Sicherheitseinstellung deaktivieren
```

Merksatz:

```
Vier-Augen-Prinzip reduziert Fehler und Missbrauch.
```

Regelmäßige Überprüfung

Sicherheitsmaßnahmen müssen regelmäßig geprüft werden.

Zu prüfen:

```
Patchstand
offene Ports
Benutzerrechte
Adminrechte
Firewall-Regeln
Backups
Restore-Tests
Logs
Zertifikate
```

```
Cloud-Freigaben
externe Zugänge
```

Merksatz:

```
Sicherheit ist ein laufender Prozess,
kein einmaliger Zustand.
```

Typische Fehler bei Patchmanagement

Häufige Fehler:

```
kein Inventar
keine Zuständigkeit
keine Priorisierung
kritische Systeme werden vergessen
Updates werden nie getestet
kein Wartungsfenster
kein Rollback-Plan
Erfolg wird nicht geprüft
Abhängigkeiten werden übersehen
Firmware wird vergessen
Container-Images bleiben alt
```

Merksatz:

```
Patchmanagement scheitert oft an Organisation,
nicht nur an Technik.
```

Typische Fehler bei Hardening

Häufige Fehler:

```
Standardpasswörter bleiben aktiv
unnötige Dienste laufen
offene Ports bleiben erreichbar
alte Protokolle bleiben erlaubt
```

Adminzugänge sind zu breit
Logs sind deaktiviert
Fehlermeldungen zeigen Details
Cloud-Speicher ist öffentlich
Testsysteme sind ungeschützt
Baseline wird nicht kontrolliert

Merksatz:

Hardening muss überprüfbar und dauerhaft gepflegt sein.

Checkliste: Schwachstellen reduzieren

Assets erfassen.
Versionen dokumentieren.
Schwachstellenmeldungen prüfen.
Kritikalität bewerten.
Patches priorisieren.
Updates testen.
Wartungsfenster planen.
Rollback vorbereiten.
Patches ausrollen.
Erfolg prüfen.
Systeme härten.
unnötige Dienste deaktivieren.
Rechte begrenzen.
Logs aktivieren.
regelmäßige Scans durchführen.
Maßnahmen dokumentieren.

Merksatz:

Schwachstellenmanagement verbindet Technik,
Prozesse
und Dokumentation.

Typische IHK-Fragen

In AP1 und AP2 kann zum Beispiel gefragt werden:

- Was ist eine Schwachstelle?
- Was ist der Unterschied zwischen Schwachstelle, Bedrohung und Risiko?
- Was ist eine CVE?
- Was ist CVSS?
- Was ist ein Exploit?
- Was ist ein Zero-Day?
- Was ist Patchmanagement?
- Warum ist Patchmanagement wichtig?
- Welche Systeme müssen gepatcht werden?
- Warum braucht man ein Inventar?
- Wie priorisiert man Patches?
- Was ist ein Notfallpatch?
- Warum braucht man ein Wartungsfenster?
- Was bedeutet Rollback?
- Was ist Hardening?
- Warum ist Standardkonfiguration nicht automatisch sicher?
- Was ist eine Baseline?
- Was bedeutet Configuration Drift?
- Was ist ein Schwachstellenscan?
- Was ist der Unterschied zwischen Schwachstellenscan und Penetrationstest?

Typische Prüfungsfallen

Schwachstelle,
Bedrohung
und Risiko nicht verwechseln.

CVE benennt bekannte Schwachstellen.

CVSS bewertet Kritikalität.

Hoher CVSS-Wert ersetzt keine eigene Risikoanalyse.

Exploit nutzt eine Schwachstelle aus.

Zero-Day hat noch keinen regulären Patch.

Patchmanagement ist ein Prozess,
nicht nur ein Klick auf Update.

Ohne Inventar kein gutes Patchmanagement.

Firmware und Netzwerkgeräte nicht vergessen.

Container-Images müssen ebenfalls aktualisiert werden.

Cloud nimmt nicht jede Patchpflicht ab.

Patches priorisieren,
nicht blind alles gleich behandeln.

Kritische Systeme brauchen Test und Rollback.

Hardening reduziert Angriffsfläche.

Standardkonfiguration ist nicht automatisch sicher.

Unnötige Dienste deaktivieren.

Alte Protokolle abschalten.

Baseline definiert sicheren Grundzustand.

Configuration Drift regelmäßig prüfen.

Schwachstellenscan findet bekannte Probleme,
aber nicht alles.

Penetrationstest ist kontrollierte Angriffssimulation.

Wichtige Begriffe kurz erklärt

Begriff	Kurze Erklärung
Schwachstelle	ausnutzbare Sicherheitslücke
Bedrohung	mögliches schädliches Ereignis

Begriff	Kurze Erklärung
Risiko	Wahrscheinlichkeit und Auswirkung eines Schadens
Schutzmaßnahme	reduziert Risiko
CVE	eindeutige Kennung bekannter Schwachstellen
CVSS	Bewertung der Kritikalität einer Schwachstelle
Exploit	Ausnutzung einer Schwachstelle
Zero-Day	Schwachstelle ohne verfügbaren regulären Patch
Patch	Korrektur oder Sicherheitsupdate
Update	kleinere Aktualisierung
Upgrade	größere Versionsänderung
Patchmanagement	organisierte Verwaltung von Updates
Inventarisierung	Erfassung vorhandener Systeme
Asset	schützenswerte Ressource
Asset Management	Verwaltung von IT-Ressourcen
Notfallpatch	schneller Patch bei kritischer Schwachstelle
Wartungsfenster	geplanter Änderungszeitraum
Rollback	Rücknahme einer Änderung
Firmware	Software auf Geräten
Dependency Management	Verwaltung von Softwareabhängigkeiten
SBOM	Stückliste von Softwarebestandteilen
Schwachstellenscan	automatisierte Suche bekannter Schwachstellen
False Positive	fälschlich gemeldetes Problem
False Negative	nicht erkannter echter Fehler
Penetrationstest	kontrollierte Angriffssimulation
Hardening	sichere Systemhärtung
Baseline	definierter sicherer Grundzustand
Configuration Drift	Abweichung von Soll-Konfiguration
Compliance-Scan	Prüfung auf Einhaltung von Vorgaben
Vier-Augen-Prinzip	Prüfung durch mindestens zwei Personen

IHK-sichere Kurzformulierung

Eine Schwachstelle ist eine ausnutzbare Sicherheitslücke in Technik, Konfiguration, Organisation oder Prozessen. Eine Bedrohung ist ein mögliches schädliches Ereignis, während Risiko die

Kombination aus Eintrittswahrscheinlichkeit und Schadensauswirkung beschreibt. Bekannte Schwachstellen werden häufig mit CVE-Kennungen beschrieben und mit CVSS nach Kritikalität bewertet. Patchmanagement umfasst das Erfassen betroffener Systeme, Bewerten von Schwachstellen, Priorisieren, Testen, Einspielen und Dokumentieren von Updates. Hardening bedeutet, Systeme sicher zu konfigurieren, unnötige Dienste zu deaktivieren, Rechte zu begrenzen, unsichere Protokolle abzuschalten und sichere Baselines einzuhalten. Schwachstellenscans und Penetrationstests helfen, Sicherheitsprobleme zu erkennen und Maßnahmen abzuleiten.

Merksätze

Schwachstelle = ausnutzbare Lücke.

Bedrohung = mögliches Schadereignis.

Risiko = Wahrscheinlichkeit und Auswirkung.

Schutzmaßnahme senkt Risiko.

CVE benennt bekannte Schwachstellen.

CVSS bewertet Kritikalität.

Exploit nutzt Schwachstelle aus.

Zero-Day hat noch keinen regulären Patch.

Patch = Korrektur oder Sicherheitsupdate.

Update kleiner,
Upgrade größer.

Patchmanagement ist ein Prozess.

Ohne Inventar kein gutes Patchmanagement.

Assets müssen bekannt sein.

Kritische Patches priorisieren.

Öffentlich erreichbare Systeme zuerst prüfen.

Notfallpatch kann außerplanmäßig nötig sein.

Wartungsfenster planen.

Rollback vorbereiten.

Patch-Erfolg prüfen.

Firmware nicht vergessen.

Netzwerkgeräte müssen aktualisiert werden.

Clients sind häufige Einstiegspunkte.

Cloud nimmt nicht jede Patchpflicht ab.

Container-Images altern.

Abhängigkeiten sind Teil der Sicherheit.

SBOM zeigt Softwarebestandteile.

Schwachstellenscan findet bekannte Probleme.

False Positives prüfen.

False Negatives bedenken.

Penetrationstest prüft Angriffswege.

Hardening reduziert Angriffsfläche.

Standardkonfiguration ist nicht automatisch sicher.

Unnötige Dienste deaktivieren.

Rechte minimieren.

Alte Protokolle abschalten.

Baseline definiert sicheren Grundzustand.

Configuration Drift vermeiden.

Sicherheit regelmäßig überprüfen.

Schwachstellenmanagement braucht Technik,
Prozess
und Dokumentation.