16.5 Webangriffe: SQL Injection, XSS, CSRF und unsichere Webanwendungen

Webanwendungen sind häufige Angriffsziele, weil sie oft öffentlich erreichbar sind und direkt mit Benutzern, Datenbanken, APIs und Backend-Systemen kommunizieren.

Typische Webangriffe sind:

- SQL Injection
- Cross-Site Scripting
- Cross-Site Request Forgery
- Directory Traversal
- File Upload Attack
- Session Hijacking
- Broken Authentication
- Broken Access Control
- unsichere API
- unsichere Fehlerausgaben
- unsichere Konfiguration

Merksatz:

Webangriffe nutzen häufig fehlerhafte Eingaben,
fehlende Zugriffskontrollen
oder unsichere Sitzungen aus.

Warum Webanwendungen gefährdet sind

Webanwendungen sind oft besonders angreifbar, weil sie:

öffentlich erreichbar sind
Benutzeranfragen verarbeiten
Datenbanken nutzen
Dateien entgegennehmen
Sitzungen verwalten
APIs bereitstellen
externe Dienste anbinden

mit sensiblen Daten arbeiten

Merksatz:

Je mehr Eingaben und Schnittstellen eine Anwendung hat,
desto größer ist die Angriffsfläche.

Eingaben als Risiko

Viele Webangriffe beginnen mit Benutzereingaben.

Beispiele für Eingaben:

Loginformular
Suchfeld
Kontaktformular
URL-Parameter
Cookie
HTTP-Header
Datei-Upload
API-Request
JSON-Daten
Formularfelder

Problem:

Die Anwendung vertraut Eingaben zu stark
oder verarbeitet sie unsicher.

Merksatz:

Benutzereingaben niemals blind vertrauen.

Client und Server unterscheiden

Client:

Browser oder App des Benutzers

Server:

Webserver,
Anwendung,
API
oder Backend-System

Wichtig:

Prüfungen im Browser allein reichen nicht aus.

Warum?

Ein Angreifer kann Browserprüfungen umgehen
und direkt eigene Anfragen an den Server senden.

Merksatz:

Sicherheitsprüfungen müssen serverseitig erfolgen.

Frontend-Validierung

Frontend-Validierung prüft Eingaben im Browser.

Beispiel:

Pflichtfeld prüfen
E-Mail-Format prüfen
Zeichenanzahl begrenzen

Vorteil:

bessere Benutzerführung

Nachteil:

```
leicht umgehbar
```

Merksatz:

```
Frontend-Validierung hilft Benutzern,
ist aber keine Sicherheitsgrenze.
```

Backend-Validierung

Backend-Validierung prüft Eingaben auf dem Server.

Sie ist sicherheitsrelevant.

Beispiele:

```
Datentyp prüfen
Wertebereich prüfen
erlaubte Zeichen prüfen
Dateityp prüfen
Berechtigung prüfen
Pflichtfelder prüfen
Länge begrenzen
```

Merksatz:

```
Sicherheitsrelevante Prüfung gehört ins Backend.
```

SQL Injection

SQL Injection ist ein Angriff auf Datenbankabfragen.

Dabei schleust ein Angreifer SQL-Code in Eingaben ein.

Ziel:

```
Daten auslesen
Daten verändern
Login umgehen
```

```
Daten löschen
Datenbankstruktur ausspähen
Rechte ausweiten
```

Merksatz:

```
SQL Injection nutzt unsicher zusammengesetzte Datenbankabfragen aus.
```

SQL Injection einfach erklärt

Eine Anwendung erwartet zum Beispiel:

```
Benutzername

und

Passwort
```

Wenn die Anwendung Eingaben unsicher direkt in SQL einfügt, kann ein Angreifer die Abfrage manipulieren.

Problem:

```
Eingabe wird nicht mehr nur als Wert behandelt,
sondern als Teil des SQL-Befehls.
```

Merksatz:

```
Bei SQL Injection wird Eingabe zu Befehl.
```

Unsichere SQL-Abfrage als Prinzip

Unsicheres Prinzip:

```
SQL-Befehl wird aus Text und Benutzereingabe zusammengebaut.
```

Beispielgedanke:

```
SELECT ...
WHERE user = Eingabe
AND password = Eingabe
```

Wenn Eingabe nicht sauber behandelt wird, kann sie die Logik der Abfrage verändern.

Merksatz:

```
SQL nicht durch ungeprüftes Zusammenkleben von Text und Eingaben bauen.
```

Folgen von SQL Injection

Mögliche Folgen:

```
Login ohne gültiges Passwort
Auslesen von Kundendaten
Auslesen von Passworthashes
Verändern von Datensätzen
Löschen von Tabellen
Manipulation von Bestellungen
Zugriff auf interne Informationen
vollständige Kompromittierung der Anwendung
```

Merksatz:

```
SQL Injection kann sehr schwere Datenbankangriffe ermöglichen.
```

Schutz vor SQL Injection

Wichtige Schutzmaßnahmen:

```
Prepared Statements
parametrisierte Abfragen
Eingabevalidierung
keine SQL-Verkettung mit Benutzereingaben
Rechte der Datenbankbenutzer begrenzen
Fehlermeldungen nicht detailliert ausgeben
```

```
Code Reviews
Web Application Firewall als zusätzliche Schicht
```

Merksatz:

```
Parametrisierte Abfragen sind die wichtigste Maßnahme gegen SQL Injection.
```

Prepared Statements

Prepared Statements trennen SQL-Befehl und Werte.

Die Datenbank erkennt:

```
Das ist der SQL-Befehl.

Das sind nur Werte.
```

Dadurch wird verhindert, dass Benutzereingaben als SQL-Code ausgeführt werden.

Merksatz:

```
Prepared Statements behandeln Eingaben als Daten,
nicht als Befehl.
```

Datenbankrechte begrenzen

Die Webanwendung sollte nicht mit einem Datenbankkonto arbeiten, das mehr Rechte hat als nötig.

Beispiel:

```
Anwendung muss nur lesen und schreiben,
aber keine Tabellen löschen.
```

Dann sollte das Datenbankkonto auch keine DROP-Rechte haben.

Merksatz:

Weniger Datenbankrechte begrenzen Schaden bei Angriffen.

Cross-Site Scripting

Cross-Site Scripting wird abgekürzt:

XSS

Bei XSS wird schädlicher Code in eine Webseite eingeschleust, der im Browser anderer Benutzer ausgeführt wird.

Meist geht es um JavaScript im Browser.

Merksatz:

XSS greift Benutzer über die Webseite und den Browser an.

XSS einfach erklärt

Eine Webseite zeigt Benutzereingaben wieder an.

Beispiel:

Kommentar
Profilname
Suchbegriff
Chatnachricht

Wenn die Eingabe nicht korrekt behandelt wird, kann ein Angreifer Skriptcode einfügen.

Dieser Code läuft dann im Browser anderer Benutzer.

Merksatz:

XSS entsteht,
wenn Eingaben ungefiltert als aktive Webseite ausgegeben werden.

Folgen von XSS

Mögliche Folgen:

```
Session-Cookies stehlen
Benutzeraktionen auslösen
Webseiteninhalt manipulieren
Phishing innerhalb echter Webseite
Tokens auslesen
Benutzer umleiten
Schadcode nachladen
Formulardaten abgreifen
```

Merksatz:

```
XSS kann echte Webseiten für Angriffe gegen Benutzer missbrauchen.
```

Stored XSS

Stored XSS bedeutet:

```
Schadcode wird dauerhaft gespeichert.
```

Beispiel:

```
Kommentar mit Schadcode wird in Datenbank gespeichert.
```

Wenn andere Benutzer die Seite öffnen, wird der Schadcode ausgeführt.

Merksatz:

```
Stored XSS bleibt gespeichert und trifft mehrere Benutzer.
```

Reflected XSS

Reflected XSS bedeutet:

```
Schadcode wird über eine Anfrage eingeschleust
und direkt in der Antwort zurückgegeben.
```

Beispiel:

```
manipulierter Link enthält Schadcode.
```

Benutzer klickt den Link, die Webseite gibt den Inhalt zurück, Browser führt ihn aus.

Merksatz:

```
Reflected XSS funktioniert oft über manipulierte Links.
```

DOM-based XSS

DOM-based XSS entsteht, wenn JavaScript im Browser unsicher mit Daten arbeitet.

Dabei muss der Schadcode nicht unbedingt vom Server direkt eingefügt werden.

Problematisch sind zum Beispiel:

```
unsichere Verarbeitung von URL-Fragmenten
unsichere DOM-Manipulation
ungeprüftes innerHTML
unsichere Client-Skripte
```

Merksatz:

```
DOM-XSS entsteht im Browser durch unsicheren JavaScript-Code.
```

Schutz vor XSS

Wichtige Schutzmaßnahmen:

```
Ausgaben korrekt escapen
Eingaben validieren
Content Security Policy
HttpOnly Cookies
Secure Cookies
SameSite Cookies
sichere Frameworks nutzen
```

keine ungeprüften HTML-Ausgaben
Code Reviews
gefährliche Funktionen vermeiden

Merksatz:

Gegen XSS ist korrektes Escaping bei der Ausgabe besonders wichtig.

Escaping

Escaping bedeutet:

Sonderzeichen werden so umgewandelt,
dass sie nicht als Code ausgeführt werden.

Beispiel:

Ein kleiner-als-Zeichen wird nicht als HTML-Tag interpretiert,
sondern als sichtbares Zeichen ausgegeben.

Merksatz:

Escaping macht aus potenziellem Code normalen Text.

Content Security Policy

Content Security Policy wird abgekürzt:

CSP

Eine CSP ist eine Sicherheitsrichtlinie im Browser.

Sie kann festlegen:

welche Skripte erlaubt sind
von welchen Quellen Inhalte geladen werden dürfen
ob Inline-Skripte erlaubt sind

ob externe Ressourcen erlaubt sind

Merksatz:

CSP begrenzt,
welche Inhalte der Browser ausführen darf.

HttpOnly Cookie

Ein HttpOnly Cookie kann nicht direkt durch JavaScript gelesen werden.

Vorteil:

erschwert Cookie-Diebstahl bei XSS

Wichtig:

HttpOnly schützt nicht vor allen XSS-Folgen,
aber reduziert ein wichtiges Risiko.

Merksatz:

HttpOnly schützt Cookies vor direktem JavaScript-Zugriff.

Secure Cookie

Ein Secure Cookie wird nur über HTTPS übertragen.

Vorteil:

Cookie wird nicht über unverschlüsselte HTTP-Verbindungen gesendet.

Merksatz:

Secure Cookies gehören zu HTTPS-Webanwendungen.

SameSite Cookie

SameSite steuert, ob Cookies bei seitenübergreifenden Anfragen mitgesendet werden.

Es hilft besonders gegen CSRF.

Mögliche Prinzipien:

```
streng
teilweise erlaubt
ohne Einschränkung
```

Merksatz:

```
SameSite begrenzt Cookie-Versand bei fremden Webseiten.
```

Cross-Site Request Forgery

Cross-Site Request Forgery wird abgekürzt:

```
CSRF
```

Bei CSRF wird ein angemeldeter Benutzer dazu gebracht, unbewusst eine Aktion in einer Webanwendung auszulösen.

Beispiel:

```
Benutzer ist im Online-Portal angemeldet.

Benutzer besucht eine bösartige Webseite.

Diese Webseite löst im Hintergrund eine Anfrage an das Online-Portal aus.
```

Merksatz:

```
CSRF missbraucht eine bestehende Anmeldung für fremde Aktionen.
```

CSRF einfach erklärt

CSRF funktioniert, weil der Browser bei Anfragen an eine Webseite oft automatisch Cookies mitsendet.

Wenn der Benutzer dort angemeldet ist, kann die Anfrage als gültig erscheinen.

Mögliche Aktionen:

```
E-Mail-Adresse ändern
Passwortänderung starten
Bestellung auslösen
Einstellung ändern
Benutzer hinzufügen
Überweisung vorbereiten
```

Merksatz:

```
CSRF nutzt automatisch mitgesendete Sitzungsinformationen aus.
```

Schutz vor CSRF

Wichtige Schutzmaßnahmen:

```
CSRF-Token
SameSite Cookies
Prüfung von Origin oder Referer
erneute Bestätigung sensibler Aktionen
MFA oder Passwortabfrage bei kritischen Aktionen
keine zustandsändernden Aktionen per GET
saubere Sessionverwaltung
```

Merksatz:

```
CSRF-Token schützen,
weil fremde Seiten den gültigen Token nicht kennen.
```

CSRF-Token

Ein CSRF-Token ist ein zufälliger Wert, der in Formularen oder Anfragen mitgesendet wird.

Die Anwendung prüft:

Stimmt der Token zur Sitzung?

Wenn nicht, wird die Aktion abgelehnt.

Merksatz:

CSRF-Token beweist,
dass die Anfrage aus der echten Anwendung kommt.

SQL Injection, XSS und CSRF vergleichen

Angriff	Ziel	Kernproblem
SQL Injection	Datenbank	Eingabe wird Teil von SQL
XSS	Browser anderer Benutzer	Eingabe wird als Skript ausgeführt
CSRF	angemeldete Sitzung	fremde Seite löst Aktion aus

Merksatz:

SQL Injection trifft Datenbank.
XSS trifft Browser.
CSRF missbraucht Sitzung.

Directory Traversal

Directory Traversal bedeutet:

Ein Angreifer versucht,
über Pfadangaben auf Dateien außerhalb des erlaubten Bereichs zuzugreifen.

Beispielprinzip:

Statt erlaubter Datei wird versucht,
über Pfadbestandteile in übergeordnete Verzeichnisse zu gelangen.

Mögliche Ziele:

Konfigurationsdateien
Passwortdateien
Quellcode
Logdateien
Systemdateien

Merksatz:

Directory Traversal nutzt unsichere Dateipfade aus.

Schutz vor Directory Traversal

Maßnahmen:

Dateinamen validieren
erlaubte Dateien per ID statt Pfad auswählen
Pfade normalisieren und prüfen
Zugriff auf festes Verzeichnis begrenzen
keine direkten Benutzereingaben als Dateipfad nutzen
Rechte des Webserver-Prozesses begrenzen

Merksatz:

Benutzer sollten keine freien Systempfade bestimmen können.

File Upload Attack

Bei einem File Upload Attack lädt ein Angreifer eine gefährliche Datei hoch.

Beispiele:

ausführbares Skript
Webshell
Malware
manipuliertes Bild
Datei mit falscher Endung
übergroße Datei

Archivbombe

Mögliche Folgen:

Codeausführung
Malware-Verteilung
Speicherüberlastung
Angriff auf andere Benutzer
Datenabfluss

Merksatz:

Datei-Uploads sind gefährlich,
wenn Inhalt,
Typ
und Speicherort nicht geprüft werden.

Schutz bei Datei-Uploads

Maßnahmen:

erlaubte Dateitypen begrenzen
Dateigröße begrenzen
MIME-Type prüfen
Dateiendung prüfen
Inhalt prüfen
Malware-Scan
Dateien außerhalb des Webroots speichern
zufällige Dateinamen verwenden
Ausführrechte verhindern
Uploads getrennt verarbeiten

Merksatz:

Hochgeladene Dateien nie ungeprüft ausführbar speichern.

Webshell

Eine Webshell ist ein hochgeladenes oder eingeschleustes Skript, mit dem Angreifer Befehle auf einem Server ausführen können.

Risiko:

```
vollständige Serverübernahme
Datenabfluss
weitere Angriffe
Backdoor
```

Merksatz:

```
Webshell = Fernsteuerung über Webskript.
```

Broken Authentication

Broken Authentication bedeutet:

```
Die Anmeldung oder Sitzungsverwaltung ist unsicher.
```

Beispiele:

```
schwache Passwortrichtlinien
kein MFA
Session läuft zu lange
Session-ID wird nicht erneuert
unsichere Passwortzurücksetzung
Tokens werden unsicher gespeichert
Brute Force nicht begrenzt
```

Merksatz:

```
Broken Authentication gefährdet Benutzerkonten und Sitzungen.
```

Schutz gegen Broken Authentication

Maßnahmen:

MFA
starke Passwortrichtlinien
Passwortmanager unterstützen
Rate Limiting
sichere Passwortzurücksetzung
Session-Rotation nach Login
kurze Sitzungslaufzeiten je nach Risiko
sichere Token-Speicherung
Login-Monitoring

Merksatz:

Anmeldung und Sitzungen müssen besonders geschützt werden.

Broken Access Control

Broken Access Control bedeutet:

Benutzer können auf Funktionen oder Daten zugreifen,
für die sie keine Berechtigung haben.

Beispiele:

Benutzer sieht fremde Rechnung
normaler Benutzer ruft Adminfunktion auf
URL-ID wird geändert und fremde Daten erscheinen
API gibt mehr Daten zurück als erlaubt
versteckte Schaltfläche reicht als Schutz

Merksatz:

Zugriffskontrolle muss serverseitig geprüft werden.

IDOR

IDOR steht für:

```
Insecure Direct Object Reference
```

Dabei kann ein Benutzer durch Änderung einer ID auf fremde Objekte zugreifen.

Beispiel:

```
/rechnung/1001
```

wird geändert zu:

```
/rechnung/1002
```

Wenn die Anwendung nicht prüft, ob diese Rechnung zum Benutzer gehört, liegt ein Fehler vor.

Merksatz:

```
IDOR ist fehlende Zugriffskontrolle bei direkten Objekt-IDs.
```

Schutz gegen Broken Access Control

Maßnahmen:

```
serverseitige Berechtigungsprüfung
Rollen- und Rechtekonzept
Zugriff pro Objekt prüfen
keine Sicherheit nur über versteckte Buttons
Standardmäßig Zugriff verweigern
Adminfunktionen getrennt schützen
API-Berechtigungen prüfen
Tests mit verschiedenen Benutzerrollen
```

Merksatz:

```
Jeder Zugriff muss auf dem Server autorisiert werden.
```

Unsichere API

APIs sind Schnittstellen zwischen Anwendungen.

Risiken:

```
fehlende Authentifizierung
fehlende Autorisierung
zu viele Daten in Antwort
keine Rate Limits
unsichere Tokens
unsichere CORS-Konfiguration
fehlende Eingabeprüfung
detaillierte Fehlermeldungen
```

Merksatz:

```
APIs brauchen dieselben Sicherheitsprüfungen wie Weboberflächen.
```

CORS

CORS steht für:

```
Cross-Origin Resource Sharing
```

CORS steuert, welche fremden Webseiten im Browser auf eine API zugreifen dürfen.

Unsichere CORS-Konfigurationen können gefährlich sein, wenn zu viele Ursprünge erlaubt werden.

Merksatz:

```
CORS regelt Browserzugriffe zwischen verschiedenen Ursprüngen.
```

Rate Limiting bei APIs

APIs sollten Anfragen begrenzen.

Schutz gegen:

```
Brute Force
Scraping
```

```
DDoS auf Anwendungsebene
Missbrauch von API-Schlüsseln
unbeabsichtigte Überlastung
```

Merksatz:

```
APIs brauchen Rate Limits gegen Missbrauch und Überlastung.
```

Unsichere Fehlerausgaben

Fehlermeldungen können Angreifern helfen.

Problematische Informationen:

```
Datenbankfehler
Pfade
Stacktraces
Versionsnummern
interne IP-Adressen
verwendete Frameworks
SQL-Abfragen
geheime Konfigurationen
```

Merksatz:

```
Fehlermeldungen für Benutzer knapp,
Details nur intern loggen.
```

Security Headers

Security Headers sind HTTP-Header, die Browser-Sicherheitsfunktionen steuern.

Beispiele:

```
Content-Security-Policy
Strict-Transport-Security
X-Content-Type-Options
Referrer-Policy
```

```
X-Frame-Options oder frame-ancestors
```

Merksatz:

```
Security Headers härten Webanwendungen im Browser.
```

HSTS

HSTS steht für:

```
HTTP Strict Transport Security
```

HSTS weist Browser an, eine Webseite nur per HTTPS aufzurufen.

Vorteil:

```
Schutz vor Downgrade auf HTTP
```

Merksatz:

```
HSTS erzwingt HTTPS im Browser.
```

Clickjacking

Clickjacking bedeutet:

```
Benutzer werden dazu gebracht,
auf etwas zu klicken,
ohne zu erkennen,
was sie tatsächlich anklicken.
```

Oft wird eine echte Webseite unsichtbar oder überlagert eingebettet.

Schutz:

```
X-Frame-Options
Content-Security-Policy frame-ancestors
```

Merksatz:

```
Clickjacking täuscht Benutzer bei Klickaktionen.
```

Unsichere Konfiguration

Unsichere Webkonfigurationen sind häufig.

Beispiele:

```
Standardpasswörter
Debug-Modus aktiv
Verzeichnislisting aktiv
unnötige Dienste
alte TLS-Versionen
unsichere Cipher Suites
fehlende Security Headers
Adminoberfläche öffentlich erreichbar
Testdaten in Produktion
Beispielseiten aktiv
```

Merksatz:

```
Sichere Webanwendungen brauchen sichere Konfiguration.
```

Verzeichnislisting

Verzeichnislisting bedeutet:

```
Webserver zeigt den Inhalt eines Ordners an,
wenn keine Startdatei vorhanden ist.
```

Risiko:

```
Dateien werden sichtbar,
die nicht sichtbar sein sollten.
```

Beispiele:

```
Backups
alte Versionen
Konfigurationsdateien
Uploads
Skripte
```

Merksatz:

```
Verzeichnislisting auf Webservern deaktivieren.
```

Web Application Firewall

Web Application Firewall wird abgekürzt:

```
WAF
```

Eine WAF filtert HTTP- und HTTPS-Anfragen.

Sie kann helfen gegen:

```
SQL Injection
XSS
bekannte Angriffsmuster
bösartige Bots
verdächtige Parameter
bestimmte Protokollverstöße
```

Wichtig:

```
Eine WAF ersetzt keine sichere Programmierung.
```

Merksatz:

```
WAF ist zusätzliche Schutzschicht,
kein Ersatz für sicheren Code.
```

Secure Coding

Secure Coding bedeutet:

> Software wird von Anfang an sicher entwickelt.

Dazu gehören:

> Eingabevalidierung
> Ausgabe-Escaping
> parametrisierte Datenbankabfragen
> sichere Authentifizierung
> Zugriffskontrolle
> sichere Fehlerbehandlung
> sichere Sessionverwaltung
> sichere Dateiverarbeitung
> Code Reviews
> Sicherheitstests

Merksatz:

> Sicherheit muss in der Entwicklung mitgedacht werden.

Code Review

Code Review bedeutet:

> Quellcode wird von anderen Personen geprüft.

Ziele:

> Fehler finden
> Sicherheitsprobleme erkennen
> Qualität verbessern
> Wissen teilen
> Standards einhalten

Merksatz:

```
Code Reviews helfen,
Sicherheitsfehler früh zu finden.
```

Penetrationstest

Ein Penetrationstest ist eine geplante Sicherheitsprüfung, bei der Systeme kontrolliert auf Schwachstellen getestet werden.

Wichtig:

```
klarer Auftrag
definierter Umfang
erlaubter Zeitraum
Dokumentation
keine unkontrollierte Beschädigung
Bericht mit Maßnahmen
```

Merksatz:

```
Penetrationstest ist kontrollierte Angriffssimulation.
```

Schwachstellenscan

Ein Schwachstellenscan sucht automatisiert nach bekannten Schwachstellen.

Er prüft zum Beispiel:

```
veraltete Software
offene Ports
bekannte CVEs
unsichere Dienste
fehlende Patches
schwache TLS-Konfiguration
```

Merksatz:

```
Schwachstellenscan findet bekannte Probleme,
ersetzt aber keinen vollständigen Penetrationstest.
```

OWASP

OWASP steht für:

```
Open Worldwide Application Security Project
```

OWASP ist bekannt für Sicherheitswissen zu Webanwendungen, zum Beispiel Listen häufiger Webrisiken.

Für Prüfungen ist wichtig:

```
Webanwendungen brauchen systematische Sicherheitsmaßnahmen.
```

Merksatz:

```
OWASP steht für praxisnahes Webanwendungs-Sicherheitswissen.
```

Typische Webschutzmaßnahmen

Wichtige Maßnahmen:

```
Eingaben validieren
Ausgaben escapen
Prepared Statements
CSRF-Token
sichere Cookies
Security Headers
HTTPS
HSTS
MFA
Rate Limiting
Rollen- und Rechteprüfung
sichere Datei-Uploads
sichere Fehlerbehandlung
Logging
Monitoring
Code Reviews
WAF
```

Patchmanagement

Merksatz:

Websicherheit braucht sichere Entwicklung,
sichere Konfiguration
und Überwachung.

Typische IHK-Fragen

In AP1 und AP2 kann zum Beispiel gefragt werden:

- Was ist SQL Injection?
- Wie schützt man sich vor SQL Injection?
- Was sind Prepared Statements?
- Was ist Cross-Site Scripting?
- Was ist der Unterschied zwischen Stored XSS und Reflected XSS?
- Wie schützt man sich vor XSS?
- Was ist Cross-Site Request Forgery?
- Wie schützen CSRF-Token?
- Was ist Directory Traversal?
- Warum sind Datei-Uploads gefährlich?
- Was ist eine Webshell?
- Was bedeutet Broken Authentication?
- Was bedeutet Broken Access Control?
- Was ist IDOR?
- Warum muss Zugriff serverseitig geprüft werden?
- Was ist CORS?
- Warum brauchen APIs Rate Limiting?
- Was ist HSTS?
- Was ist Clickjacking?
- Was ist eine WAF?
- Warum ersetzt eine WAF keine sichere Programmierung?

Typische Prüfungsfallen

Frontend-Validierung ist keine Sicherheitsgrenze.

Sicherheitsprüfung gehört ins Backend.

SQL Injection betrifft Datenbankabfragen.

Prepared Statements schützen gegen SQL Injection.

XSS betrifft Browser anderer Benutzer.

XSS-Schutz braucht korrektes Escaping.

Stored XSS wird gespeichert.

Reflected XSS kommt über Anfrage zurück.

DOM-XSS entsteht im Browser.

CSRF missbraucht bestehende Anmeldung.

CSRF ist nicht dasselbe wie XSS.

SameSite Cookies helfen gegen CSRF.

Directory Traversal nutzt Dateipfade aus.

Datei-Uploads nie ungeprüft ausführbar speichern.

Webshell kann Server fernsteuerbar machen.

Broken Authentication betrifft Anmeldung und Sitzung.

Broken Access Control betrifft Berechtigungen.

Versteckte Buttons sind keine Zugriffskontrolle.

IDOR entsteht durch fehlende Objektprüfung.

APIs brauchen Authentifizierung,

Autorisierung
und Rate Limits.

Fehlerdetails nicht öffentlich anzeigen.

HSTS erzwingt HTTPS.

WAF ersetzt keinen sicheren Code.

Wichtige Begriffe kurz erklärt

Begriff	Kurze Erklärung
Webangriff	Angriff auf Webanwendung oder API
Eingabevalidierung	Prüfung von Benutzereingaben
Frontend	Browser- oder Client-Seite
Backend	Server-Seite
SQL Injection	Einschleusen von SQL-Code
Prepared Statement	getrennte SQL-Abfrage mit Parametern
XSS	Cross-Site Scripting
Stored XSS	dauerhaft gespeichertes XSS
Reflected XSS	XSS über zurückgespiegelte Anfrage
DOM-XSS	XSS durch Browser-DOM-Verarbeitung
Escaping	Umwandeln von Sonderzeichen
CSP	Content Security Policy
HttpOnly	Cookie nicht per JavaScript lesbar
Secure Cookie	Cookie nur über HTTPS
SameSite	Cookie-Regel für fremde Seiten
CSRF	Cross-Site Request Forgery
CSRF-Token	Schutzwert gegen CSRF
Directory Traversal	Zugriff über manipulierte Pfade
File Upload Attack	Angriff über hochgeladene Datei
Webshell	Webskript zur Fernsteuerung
Broken Authentication	unsichere Anmeldung oder Sitzung
Broken Access Control	fehlerhafte Zugriffskontrolle

Begriff	Kurze Erklärung
IDOR	Zugriff über unsichere Objekt-ID
API	Programmierschnittstelle
CORS	Cross-Origin Resource Sharing
Rate Limiting	Begrenzung von Anfragen
Security Headers	HTTP-Sicherheitsheader
HSTS	HTTPS-Erzwingung
Clickjacking	Täuschung bei Klickaktionen
WAF	Web Application Firewall
Secure Coding	sichere Softwareentwicklung
Code Review	Prüfung von Quellcode
Penetrationstest	kontrollierte Angriffssimulation
Schwachstellenscan	automatisierte Suche bekannter Schwachstellen
OWASP	Organisation für Webanwendungssicherheit

IHK-sichere Kurzformulierung

Webangriffe richten sich gegen Webanwendungen, APIs, Sitzungen, Eingaben und Datenbanken. SQL Injection entsteht, wenn Benutzereingaben unsicher in Datenbankabfragen eingebaut werden; Schutz bieten Prepared Statements und parametrisierte Abfragen. Cross-Site Scripting schleust Skriptcode in Webseiten ein, der im Browser anderer Benutzer ausgeführt wird; Schutz bieten korrektes Escaping, sichere Frameworks, Content Security Policy und sichere Cookies. Cross-Site Request Forgery missbraucht eine bestehende Anmeldung, um ungewollte Aktionen auszulösen; Schutz bieten CSRF-Token und SameSite Cookies. Weitere Risiken sind unsichere Datei-Uploads, Directory Traversal, Broken Authentication, Broken Access Control, unsichere APIs und unsichere Konfiguration. Eine WAF kann unterstützen, ersetzt aber keine sichere Programmierung.

Merksätze

Webanwendungen sind häufig öffentlich erreichbar.

Benutzereingaben niemals blind vertrauen.

Frontend-Prüfung ist keine Sicherheitsgrenze.

Sicherheitsprüfung gehört ins Backend.

SQL Injection macht Eingabe zu SQL-Befehl.

Prepared Statements trennen Befehl und Wert.

Datenbankrechte begrenzen.

XSS führt Skript im Browser aus.

Stored XSS wird gespeichert.

Reflected XSS kommt über Anfrage zurück.

DOM-XSS entsteht im Browser.

Escaping macht Code zu Text.

CSP begrenzt ausführbare Inhalte.

HttpOnly schützt Cookies vor JavaScript-Zugriff.

Secure Cookie nur über HTTPS.

SameSite hilft gegen CSRF.

CSRF missbraucht bestehende Anmeldung.

CSRF-Token schützt vor fremden Aktionen.

SQL Injection,
XSS
und CSRF nicht verwechseln.

Directory Traversal nutzt Pfade aus.

Datei-Uploads streng prüfen.

Webshell bedeutet Fernsteuerungsrisiko.

Broken Authentication betrifft Anmeldung.

Broken Access Control betrifft Rechte.

IDOR ist fehlende Objektberechtigung.

APIs brauchen dieselbe Sicherheit wie Weboberflächen.

CORS nicht zu breit erlauben.

Rate Limiting schützt APIs.

Fehlerdetails nicht öffentlich anzeigen.

Security Headers härten Browser-Schutz.

HSTS erzwingt HTTPS.

Clickjacking täuscht Klicks.

WAF ist Zusatzschutz.

WAF ersetzt keinen sicheren Code.

Secure Coding früh einplanen.

Code Reviews finden Fehler früh.

Schwachstellenscan findet bekannte Probleme.

Penetrationstest prüft kontrolliert tiefer.