Sicherheitsmechanismen in (deinen?) Programmen

Agenda

- Einführung in die Programmierungssicherheit
- Häufige Sicherheitslücken und –risiken
- Implementierung von Sicherheitsmechanismen
- Praktische Übungen
- Abschlussdiskussion

Diskussion

- Welche Sicherheitsrisiken in Webanwendungen kennt ihr bereits?
- Welche Erfahrungen habt ihr mit Sicherheitsproblemen gemacht?
- Welche Schutzmaßnahmen habt ihr bereits implementiert?

OWASP Top 10 - Überblick

- Das Open Worldwide Application Security Project ist eine Non-Profit-Organisation mit dem Ziel, die Sicherheit von Anwendungen, Diensten und Software im Allgemeinen zu verbessern.
- Die OWASP Top 10 umfasst eine weithin anerkannte Liste der kritischsten Sicherheitsrisiken für Webanwendungen. Die Liste dient als Leitfaden für Entwickler, Sicherheitsexperten und Organisationen zur Priorisierung ihrer Bemühungen bei der Identifizierung und Minderung kritischer Sicherheitsrisiken für Webanwendungen.
- Introduction - OWASP Cheat Sheet Series

OWASP Top 10 - 2023

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery

OWASP Top 10 - 2023

Die zehn kritischsten Sicherheitsrisiken für Webanwendungen (2023)

1. Broken Access Control

„Unzureichende Zugriffsrechteprüfungen“

- Unberechtigter Zugriff auf Funktionen/Daten
- Manipulation von Benutzerrechten
- Beispiel: Zugriff auf Admin-Funktionen durch URL-Manipulation

Top 5 Schwachstellen 2020: Broken Access Control | usd AG

OWASP Top 10 - 2023

Die zehn kritischsten Sicherheitsrisiken für Webanwendungen (2023)

2. Cryptographic Failures

Kryptografische Fehlfunktionen sind Sicherheitslücken, die auftreten können, wenn kryptografische Algorithmen, Protokolle oder Implementierungen falsch verwendet werden. Dadurch können sensible Daten wie Passwörter, Kreditkartennummern und persönliche Identifikationsnummern für Angreifer zugänglich werden.

Es gibt viele verschiedene Ursachen für kryptografische Fehler. Zu den häufigsten gehören:

- **Keine Verschlüsselung verwendet** – Sensible Daten sollten immer verschlüsselt werden, wenn sie über ein Netzwerk übertragen oder auf einem Server gespeichert werden.
- **Unsichere kryptografische Funktionen** – Kryptografische Funktionen können veraltet, unsicher oder unsicher implementiert werden, was zu Schwachstellen führt.
- **Fehlerhaftes Schlüsselmanagement** – Kryptografische Schlüssel sind essenziell für die Sicherheit eines Systems, daher ist es wichtig, sie ordnungsgemäß zu verwalten.

OWASP Top 10 - 2023

Die zehn kritischsten Sicherheitsrisiken für Webanwendungen (2023)

3. Injection

„Einfügen schädlichen Codes in Anwendungen“

Beispielsweise:

- SQL-Injection
- Cross-Site Scripting (XSS) Was ist Cross Site Scripting (XSS) und wie funktioniert es? | Barracuda Networks
- Command InjectionA Pentester’s Guide to Command Injection | Cobalt
- Beispiel: SELECT * FROM users WHERE username = 'eingabe' AND password = 'passwort'

OWASP Top 10 - 2023

Die zehn kritischsten Sicherheitsrisiken für Webanwendungen (2023)

4. Insecure Design

“An example of insecure design is an app that produces overly detailed error messages. If it reports on error conditions in too much detail and offers diagnostic clues about the application environment, or other associated data, it could be revealing potentially useful information to attackers. They could then use it to launch other attacks like path transversal or SQL injection.”

Q:A04 Insecure Design - OWASP Top 10:2021

OWASP Top 10 - 2023

Die zehn kritischsten Sicherheitsrisiken für Webanwendungen (2023)

5. Security Misconfiguration

- Ungepatchte Sicherheitslücken
- Standardkonfigurationen
- Ungeschützte Dateien und Verzeichnisse
- Unnötige Dienste
- Verwendung von unsicheren XML-Dateien

Beispielsweise:

Ein häufiger Fehler, den Webmaster machen, ist das Beibehalten der Standardkonfigurationen eines CMS (Content Management System). Obwohl CMS-Anwendungen benutzerfreundlich sind, können sie Sicherheitsrisiken für Endbenutzer darstellen. Viele Angriffe sind vollständig automatisiert und zielen darauf ab, Standardkonfigurationen auszunutzen, weshalb eine Anpassung dieser Einstellungen während der CMS-Installation entscheidend ist, um eine Vielzahl potenzieller Angriffe zu verhindern.Das Anpassen von Einstellungen für Kommentare, Benutzerzugriff, Sichtbarkeit von Benutzerinformationen und Standard-Dateiberechtigungen kann die Sicherheit erheblich verbessern.

OWASP Top 10 - 2023

Die zehn kritischsten Sicherheitsrisiken für Webanwendungen (2023)

6. Vulnerable and Outdated Components

- Selbst die einfachsten Websites haben zahlreiche **Abhängigkeiten** wie Frameworks, Bibliotheken, Erweiterungen und Plugins
  - und **jede einzelne davon muss regelmäßig aktualisiert werden**. Angreifer suchen gezielt nach Websites mit **verwundbaren Komponenten**, die sie ausnutzen können, um **Malware zu verbreiten, Phishing-Angriffe durchzuführen** und mehr. Daher ist es **keine gute Idee, Updates auszulassen** – aus welchem Grund auch immer.

OWASP Top 10 - 2023

Die zehn kritischsten Sicherheitsrisiken für Webanwendungen (2023)

7. Identification and Authentication Failures

Fehlgeschlagene Authentifizierungs- und Identitätsverwaltungsmechanismen setzen Anwendungen dem Risiko aus, dass **bösartige Akteure sich als legitime Benutzer ausgeben**. Beispielsweise kann eine **Sitzungs-ID ohne Ablaufdatum unbegrenzt aktiv bleiben**. Zudem sind **schwache Passwörter anfällig für Brute-Force-Angriffe**, insbesondere wenn **keine Beschränkung der Login-Versuche** existiert.

Lösungsansätze:

- **Multi-Faktor-Authentifizierung (MFA) implementieren**
- **Passwort-Richtlinien umsetzen** (Mindestlänge, Komplexität, regelmäßige Rotation)
- **Rate-Limiting für Anmeldeversuche aktivieren**, um automatisierte Angriffe zu verhindern

OWASP Top 10 - 2023

Die zehn kritischsten Sicherheitsrisiken für **Webanwendungen (2023)**
8. Software and Data Integrity Failures

Diese Art von **Designfehler** entsteht durch die zunehmende Komplexität moderner Architekturen. Entwickler integrieren oft **Plugins und Bibliotheken aus verschiedenen Quellen**, ohne deren **Integrität zu überprüfen**. Dadurch können Angriffe auf Anwendungen erfolgen, die zu **unerlaubtem Datenzugriff, Systemkompromittierung oder Schadcode-Einschleusung führen**.

Beispiel: Log4j Schwachstelle Quelle:Kritische Schwachstelle in log4j veröffentlicht (CVE-2021-44228)

Lösungsansätze:

- **Bestandsverzeichnis aller Drittanbieter- und Open-Source-Komponenten führen**
- **Regelmäßiges Monitoring von Bedrohungen implementieren**
- **Sicherheitsüberprüfungen für externe Bibliotheken und Plugins durchführen**

OWASP Top 10 - 2023

Die zehn kritischsten Sicherheitsrisiken für **Webanwendungen (2023)**

9. Security Logging and Monitoring Failures

Schwache **Logging- und Monitoring-Funktionen** führen dazu, dass **Sicherheitsvorfälle unbemerkt bleiben**. Fehlgeschlagene Anmeldeversuche oder verdächtige Aktivitäten könnten übersehen werden, was Angreifern genug Zeit gibt, **erhebliche Schäden anzurichten**.

Lösungsansätze:

- **Alle Login-Versuche (inkl. Fehlversuche) protokollieren**
- **Backups der Log-Dateien anlegen, um Datenverlust zu vermeiden**
- **Manipulationsschutz für Logs einführen**, damit keine nachträgliche Änderung möglich ist
- **Regelmäßige Tests des Monitoring-Systems durchführen**, um Sicherheitslücken frühzeitig zu erkennen

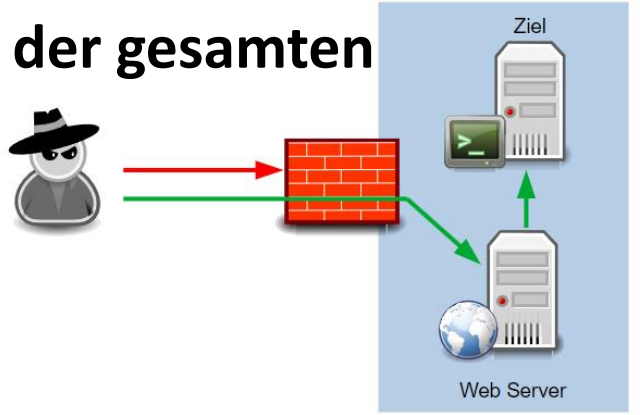
OWASP Top 10 - 2023

Die zehn kritischsten Sicherheitsrisiken für **Webanwendungen (2023)**
10. Server-Side Request Forgery

Diese Sicherheitslücke erlaubt es Angreifern, **unerlaubte Anfragen vom Server an interne oder externe Ressourcen zu senden**. Dabei manipulieren Angreifer **Eingabefelder oder Parameter**, um den Server dazu zu bringen, **beliebige URLs aufzurufen** – oft **ohne Wissen des Nutzers**.

Mögliche Angriffe:

- Zugriff auf **sensible Daten**
- Kommunikation mit **internen Systemen**, um Sicherheitsbarrieren zu umgehen
- **Ausführen von Server-Aktionen**, die zu einer **Kompromittierung der gesamten Anwendung oder Infrastruktur** führen können
- Q:Server Side Request Forgery (SSRF) – Was ist das? - Securai



Bedeutung für Entwickler

- Sicherheit als kontinuierlicher Prozess
- Integration in den Entwicklungszyklus
- Auswirkungen auf:
  - Unternehmensreputation
  - Kundenvertrauen
  - Rechtliche Konsequenzen (DSGVO)
  - Finanzielle Verluste

Pause

- 10min Pause

Vertiefung - Sicherheitsmechanismen

- Beachte folgendes für die Zukunft um deine Programme sicherer zu gestallten

Authentifizierung & Autorisierung

- Authentifizierung:
  - Mehrfaktor-Authentifizierung
  - Passwort-Hashing (bcrypt, Argon2)
  - OAuth, OpenID Connect

- Autorisierung:
  - Rollenbasiertes Zugriffsmanagement
  - Prinzip der geringsten Rechte
  - Token-basierte Zugriffskontrolle (JWT)

Sichere Kommunikation

- Transport Layer Security (TLS)
  - HTTPS-Implementierung
  - Aktuelle TLS-Version (mindestens 1.2)
  - Secure Cookies (HttpOnly, Secure)

- Content Security Policy (CSP)
  - XSS-Schutz
  - Einschränkung externer Ressourcen

Input-Validierung

Grundprinzipien:

- Whitelist statt Blacklist
- Validierung auf Server- **und** Clientseite
- Typisierung und Formatüberprüfung

[Falconbyte.net :: Java MySQL Prepared Statement (Tutorial)]

```java
// Unsicher
String query = "SELECT * FROM users WHERE name = '" + request.getParameter("name") + "'";

// Sicher mit Prepared Statements
String sql = "SELECT * FROM users WHERE name = ?";
PreparedStatement stmt = connection.prepareStatement(sql);
stmt.setString(1, request.getParameter("name"));
```

Secure Coding Practices

- Best Practices:
  - SOLID-Prinzipien für sicheren Code
  - Regelmäßige Code-Reviews
  - Statische Code-Analyse(Prozess der Analyse von Quellcode, um Fehler zu finden und die Qualität des Codes zu bewerten, ohne ihn ausführen zu müssen)

- Frameworks & Libraries:
  - Nutzung etablierter, gepflegter Bibliotheken
  - Regelmäßige Updates
  - Dependency-Scanning

Secure Software Development Lifecycle

- Security by Design
- Threat Modeling(Analyse der eigenen Software, der umliegenden Systeme und Prozesse)
- Automatic Security Testing
  - SAST (Static Application Security Testing)
  - DAST (Dynamic Application Security Testing)
  - Penetrationstests

Workshop - Secure Coding

Übungsaufgabe : Injection Prevention

Nutze folgende Anleitung :

FIAE/Unterrichtsmaterialien/Datenschutz/sql-injection-workshop.md at main · ChristopherSchwerdt/FIAE · GitHub