



openHPI – Sicherheit im Internet Angriffe im World Wide Web – Angriffsziel Web-Server

Prof. Dr. Christoph Meinel
Hasso-Plattner-Institut, Potsdam

Einführung: Angriffe im Web

Das World Wide Web (WWW) ist der am meisten genutzte Dienst im Internet und damit das für Cyberkriminelle attraktivste Angriffsziel

Das Web wird auf zwei Ebenen angegriffen:

- Über den **Web-Browser** auf der Nutzerseite des Webs
- Über den **Web-Server** auf der Seite der Dienstanbieter im Web

Angebote von Dienstaniestern im Web werden über deren **Web-Server** angeboten und ausgeliefert

Ziel von Angriffen auf Web-Server:

- bieten Angreifern Möglichkeit zur **schnellen** und **weiten** Verbreitung ihrer bösartigen Inhalte
- bieten Zugriff auf **Daten vieler Nutzer**

Angreifer nutzen dabei **Unerfahrenheit** oder **Unachtsamkeit** der Administratoren und Entwickler von Webdiensten

- Insbesondere nutzen Angreifer **Schwachstellen in der Server-Software** aus, z.B.
 - HTTP-Server-Software
 - serverseitige Skript-Interpreter (PHP, Ruby, Python, ...)
 - Schwachstellen in Webseiten und Webanwendungen

Angriffe auf Web-Server: Top 10 Schwachstellen im Web (OWASP)

OWASP – Open Web Application Security Project

listet regelmäßig Top-Schwachstellen im Web auf:

■ Design- und Programmierfehler

Platz 1: **Injektion**

Platz 2: *Fehlerhafte Authentifizierung*

Platz 3: **Cross-Site Scripting (XSS)**

Platz 4: *Unsichere Objektreferenzierungen*

Platz 6: *Preisgabe sensibler Daten*

Platz 7: *Fehlende Zugriffskontrolle nach Funktionalität*

Platz 8: **Cross-Site Request Forgery (CSRF)**

■ Administrations- und Konfigurationsfehler

Platz 5: *Fehlerhafte Sicherheitskonfiguration*

Platz 9: *Nutzung von Komponenten mit Sicherheitslücken*

Platz 10: *Ungeprüfte Weiterleitungen*



OWASP
The Open Web Application Security Project
<http://www.owasp.org>

Injection bezeichnet das **Einschleusen von Kommandos über (eigentlich) harmlose Nutzereingaben**

- Nutzereingaben werden oft ungeprüft direkt als **Parameter** in vorgefertigten Kommandos des Programmcodes verwendet
- Werden Eingaben nicht ordnungsgemäß geprüft, können Angreifer in diese Parameter **eigene Kommandos** einbauen, die vom angegriffenen Dienst dann ausgeführt werden
- Immer möglich, wenn für Nutzereingaben keine ordnungsgemäße Filterung implementiert ist und der Nutzereingabe blind vertraut wird
- Häufigste Variante im Web: → **SQL-Injection**
 - Einschleusen von Kommandos in SQL-Abfragen einer dynamischen Webanwendung
 - Angreifer kann Authentifizierung umgehen und/oder direkt Nutzerdatenbanken bearbeiten

Grundidee von Injections:

- Auf Webseite kann sich ein Nutzer ein Informationsblatt über ein Webseitenformular schicken lassen. Die Empfängeradresse [EMPFÄNGER] wird vom Webformular gelesen.

```
SENDE_EMAIL von:'werbung@email.de' zu:'[EMPFÄNGER]'
```

- Angreifer kann dies ausnutzen und anstelle der erwarteten Empfängerangabe seine Kommandos im Empfänger-Feld angeben:

```
[EMPFÄNGER] = angreifer@email.de' UND LÖSCHE_FESTPLATTE ziel:'C'
```

- Am Ende wird folgendes ausgeführt

```
SENDE_EMAIL von:'werbung@email.de' zu:'angreifer@email.de'  
UND LÖSCHE_FESTPLATTE ziel:'C'
```

Beispiel für eine SQL-Injection: Login einer Website

```
SELECT * FROM users WHERE username = '$username'
AND password = '$password' LIMIT 1;
```

- SQL-Abfrage prüft, ob vom Nutzer eingegebene Daten **\$username** und **\$password** in dieser Kombination in der Datenbank stehen
- **Angriff:** wenn Nutzer den Text **' OR 0=0 --** im **username**-Feld eingibt, kann er die Authentifizierung umgehen !

```
SELECT * FROM users WHERE username = ' ' OR 0=0 --
AND password = ' ' LIMIT 1
```

- **0=0** trifft immer zu, womit die Zugangsbedingung (**username = ' ' OR 0=0**) erfüllt ist → SQL-Anfrage liefert **positives Ergebnis**
- **--** wird für Kommentare verwendet, Passwort und Limitierung auf ein Ergebnis werden also ignoriert

Angreifer hat **Authentifizierung umgangen !**

Gegenmaßnahmen

- Müssen hier vorwiegend von **Entwicklern** getroffen werden
- **Filtern** (zur Entschärfung) von Nutzereingaben
 - Ausfiltern eingegebener Kommandos oder Steuerzeichen,
 - z.B. Anführungszeichen (letztes Beispiel) erlaubt Ausbruch aus der Parameterumgebung
- Einschränkung der **Berechtigungen**
 - Kommandos nur mit geringst benötigten Berechtigungen ausführen
- Keine **direkte Verbindung** von Nutzereingaben und Kommandos
 - Nutzereingaben nicht als Parameter in Kommandos einbauen
- Nutzung von **Programmierbibliotheken** zur „Desinfektion“ (engl. **sanitization**) von Nutzereingaben

XSS – Cross-Site-Scripting bezeichnet Schwachstelle, bei der Angreifer Skripte – kleine Programme – in fremde Webseiten einschleust, die dann bei Nutzern dieser Webseite ausgeführt werden

- XSS ist eine Art „HTML-Injection“
- Wird nur zur Bedrohung, wenn Nutzer **Ausgabe einer Webseite** durch **seine Eingaben beeinflussen** kann
- Eingabe des Angreifers initiiert Erzeugung und Auslieferung einer neuen / veränderten **Webseite mit böartigem Skriptcode**
 - böartiger Skriptcode wird dann bei allen (anderen) Besuchern der manipulierten Website ausgeführt
- Webseiten, die **nur eigene Inhalte anzeigen** (also keinen „user-generated content“ erlauben), sind nicht betroffen

Häufig werden Webseiten mit vielen dynamischen, von Nutzern erzeugten Inhalten angegriffen, z.B.

- Blogs, Wikis, Diskussionsforen, ...
- Voraussetzung: **Nutzereingaben** werden **direkt auf Webseite übernommen** ohne zu prüfen, ob Eingaben Script-Elemente enthalten
- **Beispiel:** manipulierter Forumsbeitrag

Hallo Leute! Das ist meine erste Nachricht
im Forum. **<SCRIPT> böartiger Code </SCRIPT>**

- Eingeschleuster böartiger Code wird dann (meist unbemerkt) bei allen Nutzern **ausgeführt**, die den Forenbeitrag lesen

Bösartiger Code bei XSS-Angriffen hat typisch folgende **Ziele**:

- **Einschleusen von Schadcode**, der Schwachstellen im Browser ausnutzt, um Zugriff auf Nutzer-System zu erlangen
- **Senden persönlicher Informationen** (z.B. Cookies) an Angreifer
 - Cookies können Sitzungsdaten oder Passwörter für Webseite enthalten
 - Können dazu verwendet werden, sich unter der Identität des Opfers in Webseite einzuloggen
- **Click-Jacking**, bei dem bösartiger Skriptcode im Browser des Opfers Klicks auf Werbelinks auslöst

Schutz- und Gegenmaßnahmen

Seitens der **Betreiber / Entwickler von Webseiten**

- Wie bei SQL-Injection: Nutzereingaben stets filtern nach Steuerzeichen und Kommandos (Sanitizing)

Seitens der **Besucher von Webseiten**

- Nur Links der Hauptwebseiten folgen, die man besucht hat – mindert größtes Risiko
- Funktioniert fast immer: JavaScript deaktivieren, zumindest auf Seiten, wo es nicht benötigt wird, z.B. mit **Browser-Addons** wie NoScript, Quick JavaScript Switcher, ...
- Stets **aktuelle Updates** für sämtliche Software installieren (insbesondere: Browser), damit Schadcode keine (bekannten) Sicherheitslücken ausnutzen kann

CSRF - Cross-Site Request Forgery zielt darauf ab,

- **privilegierten Nutzer** (z.B. Administrator oder eingeloggter Nutzer) dazu zu bringen, eine **untergeschobene, unerwünschte, privilegierte Aktion** in (Web-)Anwendung durchzuführen

Methode:

- **Unterschieben** eines präparierten Links, der Aktion auslöst
- Angreifer beabsichtigen, **eigene Berechtigungen zu erhöhen** durch Übernahme der (Admin-)Rechte des Webseiten-Admins
- **Angriffspunkte** können z.B. sein
 - XSS: Einschleusen des Links auf vertrauenswürdige Webseite
 - Social Engineering: Nutzer überreden zur Aktivierung eines Links, der z.B. per Mail zugesandt wird

(Vereinfachtes) Beispiel

- Nutzer ist zum Online-Banking auf **bank.de** eingeloggt und führt gerade eine Transaktion durch
- Angreifer überredet Nutzer, eine von ihm **präparierte Webseite** zu besuchen, auf der folgendes „Bild“ platziert hat:

```

```

- Wenn (bei **bank.de** eingeloggt) Nutzer diese Webseite besucht, **lädt sein Browser das „Bild“ automatisch** und veranlasst dadurch ungewollt einen Geldtransfer von seinem Konto zum Konto des Angreifers
- Nutzer muss nichts bestätigen, bemerkt Transfer nicht einmal

Schutz- und Gegenmaßnahmen

- Seitens der **Betreiber/Entwickler von Webseiten**

- Einbauen von sogenannten CSRF-Tokens:
 - Zeichenketten, die den Links angefügt werden und sich bei jedem Webseitenbesuch ändern
 - Links ohne CSRF-Token werden nicht akzeptiert

- Seitens der **Besucher von Webseiten**

- Keinen Links von nicht-vertrauenswürdigen Personen folgen
- Stets **ausloggen aus sensiblen Diensten**, z.B. Finanzdiensten oder Shopping-Plattformen