



Cyber Security



XSS Cross Site Scripting

- 01 XSS – Grundlagen & Alltag**
- 02 Typen (Arten) und Beispiele**
- 03 Schutzmaßnahmen**
- 04 Fragen bzw. Quiz**



AGENDA

01

XSS –

Grundlagen &

Alltag



Was ist XSS?

- XSS ist die Abkürzung für Cross Site Scripting.
- Was bedeutet das?
-> Angreifer schleusen eigenen Code
auf diverse Arten in fremde Webseiten ein.
- Ursachen:
Meistens keine oder fehlerhafte Validierung von Benutzereingaben



Warum ist XSS wichtig?

- Weltweit häufige Schwachstelle (siehe QWASP)
- Folgen:
 - Ermöglicht Datendiebstahl (-Exfiltration)
 - Identitätsdiebstahl
 - Betrug
- Gefährdet Nutzer & Unternehmen
 - pers. Daten
 - Reputation



XSS im Alltag

- Betroffen:
sämtliche Web Applikationen
 - Social Media
 - Onlineshops- Foren
- Potentielle Angreifer halten Ausschau nach:
 - Formularen
 - Suchfelder
 - Kommentarfunktionen



Wie funktioniert XSS?

- Angreifer schleust Code ein.
- Das Zielsystem (ein Browser oder Webserver) führt diesen Code im Namen des Opfers aus.



Was kann XSS bewirken?

- Session Übernahme.
- Datendiebstahl in Form von Cookies oder Passwörtern
- Identitätsübernahme
- Umleitung auf Fake-Seiten (zum Phishing)
- Drive-by-Downloads



klassische Ziele für XSS

- Anmeldeformulare
- Kontaktformulare
- Kommentarfelder
- Suchfunktionen



02

Typen (Arten)

und Beispiele



XSS – Drei Haupttypen

- Reflected XSS
- Stored XSS
- DOM-based XSS



Reflected XSS

- Nutzereingabe wird sofort reflektiert
- Kein Speichern auf dem Server
- Angriff erfolgt meist über manipulierte Links
- Browser führt das eingeschleuste Skript aus
- Häufig in Suchfeldern oder Fehlermeldungen



Reflected XSS - Funktionsweise

- Opfer klickt auf präparierten Link
- Server spiegelt Eingabe in HTML zurück
- Browser führt Skript aus



Reflected XSS - Beispiel

Beispielhafte URL:

`http://domaene.de/search?q=<script>alert('XSS')</script>`

PHP-Code auf Serverseite:

```
<?php echo "Suchergebnis: " . $_GET["q"]; ?>
```



Stored XSS

- Schädlicher Code wird dauerhaft gespeichert
- Angriff wird bei jedem Seitenaufruf automatisch ausgelöst
- Klassisches Beispiel:
Kommentar mit Skript
- Höchstes Gefahrenpotenzial, besonders in Admin-Bereichen



Stored XSS

- Schädlicher Code wird dauerhaft gespeichert
- Angriff wird bei jedem Seitenaufruf automatisch ausgelöst
- Klassisches Beispiel:
Kommentar mit Skript
- Höchstes Gefahrenpotenzial, besonders in Admin-Bereichen



Stored XSS – Funktionsweise

- Angreifer schreibt Kommentar mit Skript
- Server speichert Kommentar in DB
- Browser anderer Nutzer lädt Kommentar. Folge:
Das Skript wird ausgeführt.



Stored XSS – Beispiel

HTML-Code auf dem Webserver (Auszug...):

```
<form method="POST">  
  <textarea name="comment">  
    <input type="submit" value="Absenden">  
  </textarea>  
</form>
```



Stored XSS – Beispiel

Serverseitiges PHP-Script (unsicheres Speichern & Anzeigen):

```
// Speichern (vereinfacht):  
file_put_contents("comments.txt", $_POST["comment"], FILE_APPEND);  
  
// Anzeige:  
echo file_get_contents("comments.txt");
```

Eingabe durch Angreifer:

```
<script>alert('Quae demonstranda erant!')</script>
```



DOM-Based XSS

- Angriff erfolgt komplett im Browser
- Manipuliertes DOM durch JavaScript (z. B. via document.location)
- Kein Einfluss auf Serverantwort
- Schwer zu entdecken durch klassische Scanner



DOM-Based Beispiel (unsicher)

HTML-Code / Seite 1 von 2

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM XSS Beispiel</title>
</head>
<body>
  <h1>Testseite zur Demonstration von DOM-XSS</h1>
  <p id="output"></p>

  <script>
    // "name"-Parameter aus der URL holen:
    const params = new URLSearchParams(window.location.search);
    const name = params.get("name");
```



DOM-Based Beispiel (unsicher)

HTML-Code / Seite 2 von 2

```
// Namen direkt in das DOM einfügen (unsicher!)  
document.getElementById("output").innerHTML = "Hallo, " + name + "!";  
</script>  
</body>  
</html>
```



DOM-Based Beispiel (unsicher)

Angriff:

Die Seite (URL) wird mit dem script-Tag als Parameter aufgerufen:

`vulnerable.html?name=<script>alert('DOM XSS Test!')</script>`

Ergebnis:

Das Skript wird ausgeführt, weil der Input ungefiltert in innerHTML eingefügt wird



DOM-Based Beispiel

sicherere Variante:

Vermeidung der Ausführung durch sichere Manipulation des DOM
mit ,textContent':

```
document.getElementById("output").textContent = "Hallo, " + name + "!";>
```



XSS weiterführende Informationen bzw. CTF

Der folgende Link ist eine wertvolle Resource, um das Wissen zu vertiefen.

<https://portswigger.net/web-security/cross-site-scripting>



AGENDA

03

Schutz- maßnahmen



Content Security Policy (CSP)

Definition:

Die Content Security Policy ist eine Sicherheitsrichtlinie, die Webentwickler per HTTP-Header oder Meta-Tag setzen können, um unerwünschte Inhalte zu blockieren.

Zum Beispiel, um Cross-Site Scripting (XSS) und Code-Injektionen zu verhindern.



Schutz durch Entwickler (Code-Ebene)

Der wichtigste Punkt:

Entwickler müssen immer kontextsensitiv escapen oder encodieren, je nachdem, wo die Benutzereingabe landet:

- im HTML-Text,
- im Attribut,
- in JavaScript,
- in der URL

Dabei helfen Frameworks wie React, Angular oder Django, die standardmäßig escapen – wenn man ihre Mechanismen nicht umgeht.

Außerdem sollte nie auf Client-Seite validiert werden – nur zusätzlich.



Schutz durch HTTP-Header & Browser

- Content Security Policy (CSP):
Beschränkt, welche Skripte geladen oder ausgeführt werden dürfen.
Sehr mächtig!
- X-XSS-Protection:
älterer Schutzmechanismus (mittlerweile veraltet).
- Strict MIME Type Checking:
Verhindert das Ausführen von HTML oder JS, wenn der Content-Type falsch ist.



Schutz durch Testing & Tools

- Klassische Tools nutzen, wie:
 - Burp Suite
 - OWASP ZAP
 - Netsparker
- einfache CURL-Requests
- Manche Browser-Plugins können ggf. helfen Schwachstellen aufzudecken.



AGENDA

04

Fragen bzw. Quiz



Quiz

Frage 1

Was ist das Hauptziel von XSS?



Quiz

Antwort zu Frage 1:

Einschleusen und Ausführen von fremdem Code.



Quiz

Frage 2

Wo kann XSS auftreten ?



Quiz

Antwort zu Frage 2:

**In Formularen, Kommentaren,
URLs, Suchfeldern.**



Quiz

Frage 3

Welche Typen von XSS gibt es?



Quiz

Antwort zu Frage 3:

**Reflected, Stored und
DOM-based**



Schlusswort:

**"XSS ist kein Bug – es ist das
Ergebnis eines fehlenden
Sicherheitsdenkens."**



DANKE!

Gibt es noch Fragen?





CloudCommand