



Hochschule
Albstadt-Sigmaringen

Albstadt-Sigmaringen University

Web Security



Dipl. Ing. Sven Eppler (FH)
sodge IT GmbH

A white speech bubble with a dark blue background. The bubble has a rectangular body and a triangular tail pointing towards the bottom-left corner.

Never trust a user!

Input Validation

- Alles was an den Client ausgeliefert wurde, gilt als unsicher
 - Formularfelder, URLs, HTML, JavaScript
 - Der Client kann alles ändern (Developer Tools)
 - Die Request können auch von einem Tool stammen (cURL)
- Daher **muss** der Server validieren
- Aus **UserExperience** gründen kann *zusätzlich* auch der **Client** validieren

Input Validation

- **Was muss validiert werden?**
 - Ist der Parameter vorhanden?
 - Datentyp?
 - Wertebereich?
 - Plausibilität?
- **Wie wird validiert?**
 - Im einfachsten Fall mit einfachen if-Statements
 - Besser: Validation-Frameworks
 - <https://express-validator.github.io/docs/index.html>

Input Validation

- **Wann wird validiert?**

- Bei jedem Request! Immer!
- Noch bevor das erste mal Daten aus dem Request verarbeitet werden

→ wenn Daten vom Nutzer gespeichert werden

- **Wie behandelt man Validierungsfehler?**

- Am einfachsten: HTTP Response 400 Bad Request
- Besser: Rückmeldung „Was ist falsch?“.
Highlighting der passenden Formulare (z.B. `error-class`)
- Verwendung ganzer Formular-Frameworks
 - Angular, React, etc. bieten umfangreiche Validierungsoptionen

Input Validation in Express

- Handgemachte Validierung

```
router.post('/entries', (req, res, next) => {  
    const {name, text} = req.body;  
  
    if(name === undefined)  
        return res.status(400).send("name param missing");  
  
    if(text === undefined)  
        return res.status(400).send("text param missing");  
  
    if(text.length() < 20)  
        return res.status(400).send("text is too short");  
  
    // Process the request now  
});
```

Input Validation in Express

- Handgemachte Validierung:
 - Sehr Fehleranfällig
 - Ermüdend da im wesentlichen immer wieder das selbe programmiert wird
 - Manchmal der einzige Ausweg, wenn das zu validierende Objekt zu kompliziert/besonders ist
 - Verletzt das DRY (Don't Repeat Yourself) Theorem

Input Validation in Express

- **Besser:**
 - Verwenden Sie ein Modul zur Validierung
z.B. Validate.js
 - Je populärer, desto besser, denn damit sinkt die Wahrscheinlichkeit für Fehler in der Validierung
 - Verwenden des Middleware-Konzeptes um die Validierung vor der Action durchzuführen

Input Validation in Express

- Verwendung von express-validator

```
const { check, validationResult } = require('express-validator');

router.post(
  '/entries',
  [
    check("name").not().isEmpty(),
    check("text").not().isEmpty(),
  ],
  (req, res, next) => {
    const errors = validationResult(req);
    if(!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }
    // Process Request
  });
```

Input Validation in Express

- **Nutzung des Middleware-Konzeptes**
 - Eine Route kann spezifisch viele Middlewares haben
 - Ein Validierer kann so Schritt für Schritt den Request validieren
 - Eine generalisierte Error-Response-Middleware könnte automatisch mit HTTP Status 400 antworten
 - Verhindert ultimativ sogar die komplette Ausführung Ihres Codes, wenn der Request nicht gültig ist!
- **Nutzung eines Validierungs Frameworks**
 - DRY Konform
 - Übersichtlicher
 - Komplexe Validierung einfach gemacht: z.B. eMail-Adresse

A white speech bubble with a dark blue background. The bubble has a rectangular body and a triangular tail pointing downwards and to the left.

Get an injection!

Injection

- Injection-Lücken sind das Nr. 1 Problem im Internet
- Gefahrenstellen:
 - Überall, wo User-Input mit anderen Strukturierten Daten zusammen geführt wird, die später ausgewertet werden:
 - HTML
 - JavaScript
 - CSS
 - SQL
 - JSON

HTML Injection

- Live-Demo „guestbookAPI“
- Böartiger Gästebucheinträge:

```
<strong>Hallo!</strong>
```

```
</div></div></div></div></div></div>Broken.
```

```
<h1>Hallo Welt!</h1>
<span style='color: red'>Dieser Eintrag ist besonders!</span><br/>
Besuch doch meine <a href='http://google.de'>Website!</a>.
<img
  src='https://media.giphy.com/media/xT0xeJpnrWC4XWblEk/giphy.gif' />
```

HTML Injection

- Was passiert?
 - Der vom User eingegeben Input wird **unverändert** ins HTML eingefügt. Dadurch entsteht ein komplett anderes oder Fehlerhaftes HTML-Dokument.
- Lösung:
 - HTML-Escaping
 - Ersetzen der speziellen HTML-Steuerzeichen durch entsprechende Ersatzzeichen:
 - `< = <`
 - `> = >`
 - `" = "`
 - `& = &`

HTML Injection

- **Input:**
 - `<h1>Hallo Welt!</h1>`
- **Escaped HTML:**
 - `<h1>Hallo Welt!</h1>`
- **Führt in der Anzeige zu:**
 - `<h1>Hallo Welt!</h1>`
- **Das angezeigte HTML ist jetzt aber kein HTML, sondern nur Text in der Webpage, der aussieht wie HTML. Wird daher vom Browser auch NICHT wie HTML interpretiert!**

HTML Injection

- Vermeidung von Injections:
 - Input auf Serverseite HTML-Escapen
 - Auf Client-Seite beim dynamischen erzeugen von HTML entweder:
 - Input ebenfalls Escapen
 - Frameworks liefern häufig passende Methoden, jQuery z.B.
`Element.text(input)` statt `DOMElement.html(input)`

Script Injection

- Überall wo HTML injected werden kann, könnte auch ein JavaScript injected werden
- Das Script läuft dann im Context des aktuellen Users
- Damit kann die komplette Applikation übernommen werden
 - AJAX-Request, Cookies auslesen, andere JS-Funktionen aufrufen, etc.

```
<script>alert("Injection!");</script>
```

SQL Injections

- Angriff auf den Datenbankserver im Backend
- Einfügen von SQL-Befehlen wo keine vorgesehen waren
- Problem: Identisch zu HTML
 - Text mit bestimmter Bedeutung wird unkontrolliert der Datenbank zum interpretieren übergeben

SQL Injections

- **Beispiel: Wetterseite**

- Eingabefeld für die Stadt
- Hinterlegtes Query:
`SELECT * FROM Weather WHERE city = "$UserInput"`
- Beispieleingabe: **Albstadt**
- Erzeugtes Query:
`SELECT * FROM Weather WHERE city = "Albstadt"`
- Schadhafte Eingabe: **" OR 1=1 --**
- Erzeugtes Query:
`SELECT * FROM Weather WHERE city = "" OR 1=1 -- "`

SQL Injection

- **Was unternimmt man dagegen?**
 - Niemals userInput direkt in SQL integrieren!
 - Verwendung von Prepared Statements und Bind Parameter
- **Prepared Statements bieten Platzhalter für Variablen an: "?"**
 - `SELECT * FROM Weather WHERE City = ?`
- **Erst später wird dem Platzhalter ein Wert gegeben**
 - `preparedStatement.run("Albstadt");`
- **Der Datenbanktreiber kümmert sich jetzt um das notwendige Escaping**

SQL Injection

- Prepared Statements Beispiel

```
// Laden und initialisieren von sqlite3
let sqlite3 = require('sqlite3');
let db = new sqlite3.Database('some.db');

// Definiere '?' als Platzhalter
let preparedStatement = db.prepare("SELECT * FROM Weather WHERE City = ?");

// Ersetze den ersten Platzhalter mit dem Wert Albstadt
// und führe das statement aus
preparedStatement.run("Albstadt");

// Fertigstellen des Statement-Objektes (auch auf DB seite)
preparedStatement.finalize();
```

SQL Injection

- Große Gefahr durch tools wie SQLMap
- Angreifer können im schlimmsten Fall die gesamte Datenbank:
 - Auslesen
 - Verändern
 - Löschen
- Häufig sind sogar Shell-Exploits und RemoteCodeExecution möglich, wenn die angegriffene Datenbank entsprechende Fehler/Features hat

Moral von der Geschichte

- User Input immer misstrauen!
- Nur nach sorgfältiger Prüfung ans System übergeben
- Besondere Vorsicht bei Shell aufrufen!
 - Remote Code Execution
- Auch Module sind gefährdet:
 - Remote Code Execution via ImageMagick
<https://imagemagick.com>
- OWASP CheatSheets
 - <https://cheatsheetseries.owasp.org/index.html>