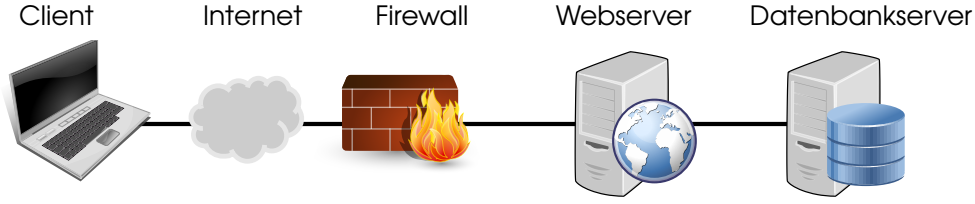
Informationssicherheit

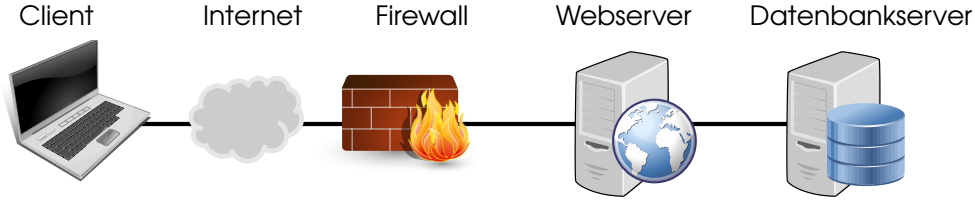
3. Schwachstellen

Prof. Dr. Christoph Skornia
christoph.skornia@oth-regensburg.de

Web-Schwachstellen



Web-Schwachstellen

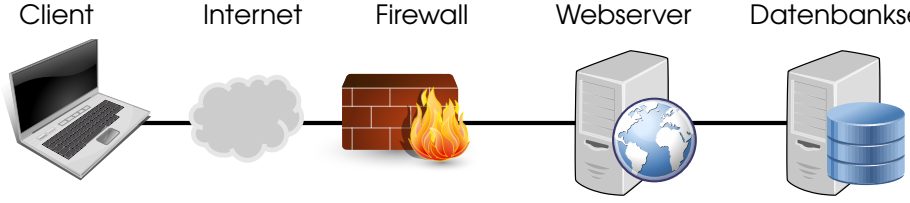


http-Anfrage ⟶

http-Antwort + aktiver Inhalt ⟵

(JavaScript, Flash, Java)

Web-Schwachstellen

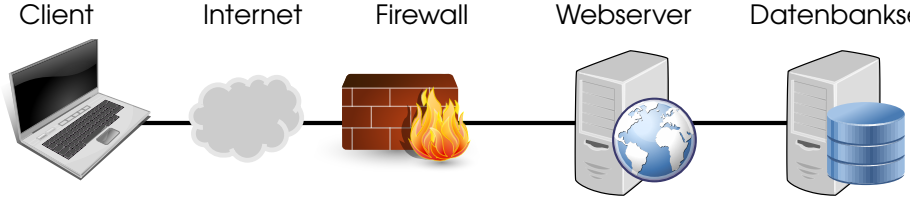


http-Anfrage ⟶
http-Antwort + aktiver Inhalt ⟵
(JavaScript, Flash, Java)

Apache
IIS

PHP
JSP
Perl
Python
Servlets
C, C++

Web-Schwachstellen



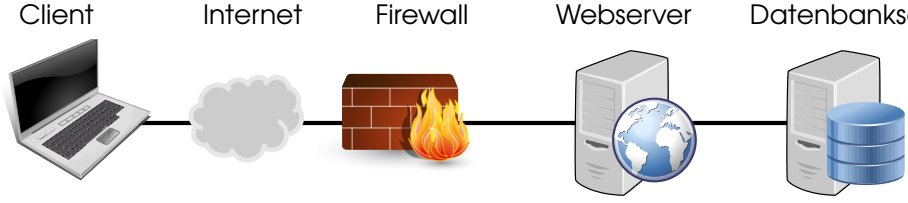
http-Anfrage ⟶
http-Antwort + aktiver Inhalt ⟵
(JavaScript, Flash, Java)

Apache
IIS

PHP
JSP
Perl
Python
Servlets
C, C++

DB-Konnektoren
JDBC
ODBC

Web-Schwachstellen



http-Anfrage ⟶
http-Antwort + aktiver Inhalt ⟵
(JavaScript, Flash, Java)

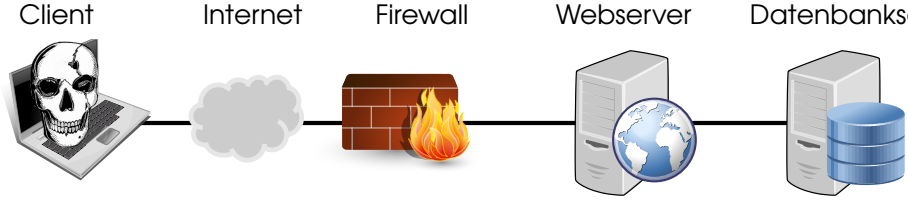
Apache
IIS

PHP
JSP
Perl
Python
Servlets
C, C++

DB-Konnektoren
JDBC
ODBC

Rich Client (Ajax, Java...):
http-Anfrage / RPC Aufruf ⟶
Antwort (JSON, XML, HTML, RPC...) ⟵

Web-Schwachstellen



http-Anfrage ⟶
http-Antwort + aktiver Inhalt ⟵
(JavaScript, Flash, Java)

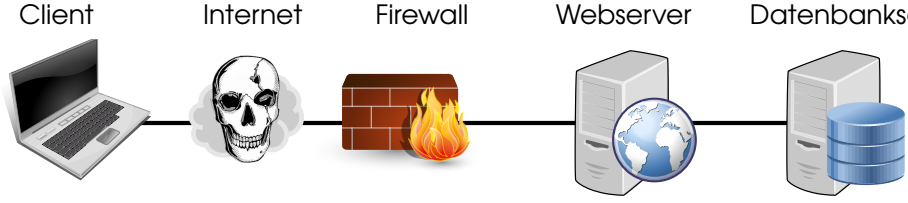
Apache
IIS

PHP
JSP
Perl
Python
Servlets
C, C++

DB-Konnektoren
JDBC
ODBC

Rich Client (Ajax, Java...):
http-Anfrage / RPC Aufruf ⟶
Antwort (JSON, XML, HTML, RPC...) ⟵

Web-Schwachstellen



http-Anfrage ⟶
http-Antwort + aktiver Inhalt ⟵
(JavaScript, Flash, Java)

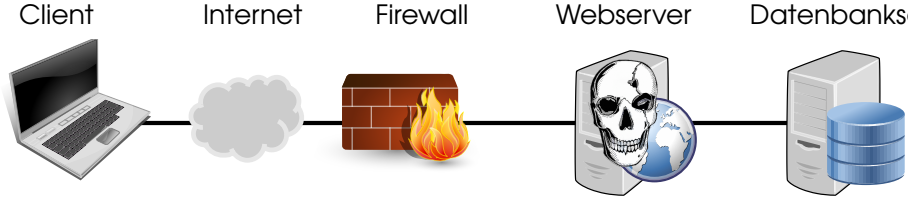
Apache
IIS

PHP
JSP
Perl
Python
Servlets
C, C++

DB-Konnektoren
JDBC
ODBC

Rich Client (Ajax, Java...):
http-Anfrage / RPC Aufruf ⟶
Antwort (JSON, XML, HTML, RPC...) ⟵

Web-Schwachstellen



http-Anfrage ⟶
http-Antwort + aktiver Inhalt ⟵
(JavaScript, Flash, Java)

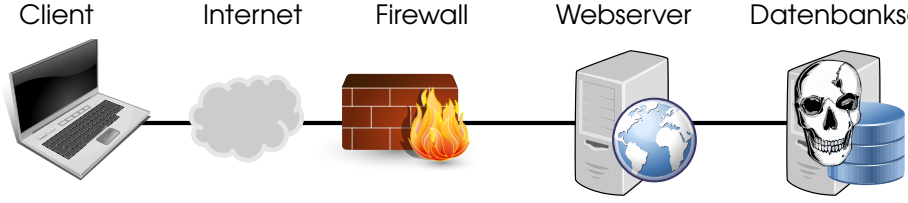
Apache
IIS

PHP
JSP
Perl
Python
Servlets
C, C++

DB-Konnektoren
JDBC
ODBC

Rich Client (Ajax, Java...):
http-Anfrage / RPC Aufruf ⟶
Antwort (JSON, XML, HTML, RPC...) ⟵

Web-Schwachstellen



http-Anfrage ⟶
http-Antwort + aktiver Inhalt ⟵
(JavaScript, Flash, Java)

Apache
IIS

PHP
JSP
Perl
Python
Servlets
C, C++

DB-Konnektoren
JDBC
ODBC

Rich Client (Ajax, Java...):
http-Anfrage / RPC Aufruf ⟶
Antwort (JSON, XML, HTML, RPC...) ⟵

http und Sessions

- http ist ein Zustandsloses Protokoll
- Cookie-Konzept:
  1. Datenstruktur vom Server erstellt, um Daten über den Client zu speichern.
  2. Häufig wird auch Authentisierungsinformation abgelegt, z.B. Hashwert(Session ID, Server-Secret)
  3. Server sendet Cookie zum Client, der ihn speichert
  4. Bei erneutem Verbindungsaufbau sendet der Client-Browser das Cookie zum Server
  5. Server prüft die darin enthaltenen Authentisierungsinfo

http und Sessions

- http ist ein Zustandsloses Protokoll
- Cookie-Konzept:
  1. Datenstruktur vom Server erstellt, um Daten über den Client zu speichern.
  2. Häufig wird auch Authentisierungsinformation abgelegt, z.B. Hashwert(Session ID, Server-Secret)
  3. Server sendet Cookie zum Client, der ihn speichert
  4. Bei erneutem Verbindungsaufbau sendet der Client-Browser das Cookie zum Server
  5. Server prüft die darin enthaltenen Authentisierungsinfo
- Beispiel:

OWASP Top Ten Web Schwachstellen

Injection

Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

Broken Authentication

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

Sensitive Data Exposure

Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

OWASP Top Ten Web Schwachstellen

XML External Entities (XXE)

Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

Broken Access Control

Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

Security Misconfiguration

Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.

OWASP Top Ten Web Schwachstellen

Cross-Site Scripting XSS

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

Insecure Deserialization

Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

OWASP Top Ten Web Schwachstellen

- **Using Components with Known Vulnerabilities**

  Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

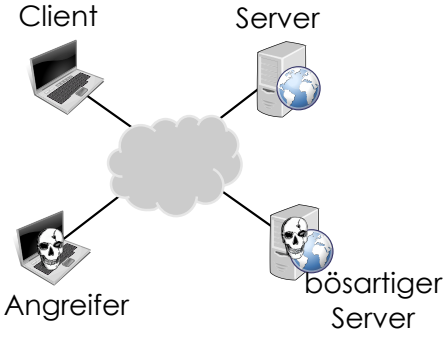
- **Insufficient Logging and Monitoring**

  Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

Cross Site Scripting (XSS)

Idee:

1. Ein bösartiger Client greift einen anderen Client an indem er ein Script auf dessen Rechner zur Ausführung bringt
2. Dieses lädt die „erbeutete" Information auf einen bösartigen Server



Drei Grundtypen:

- Stored (Type I)
- Reflected (Type II)
- DOM-Injection (Type 0)

Ursache:

- Schwachstellen im Web-Server und Ausnutzen, dass Nutzer den Inhalten, die der Browser ihnen anzeigt, vertrauen

Konsequenzen:

- Nutzer-Sitzungen können übernommen (hijacking), Web-Seiten gefälscht und missbraucht, Phishing-Angriffe durchgeführt oder aber auch der Web-Browser zur Ausführung von scriptbasierter Malware missbraucht werden.

Stored XSS (Type I)

Idee:

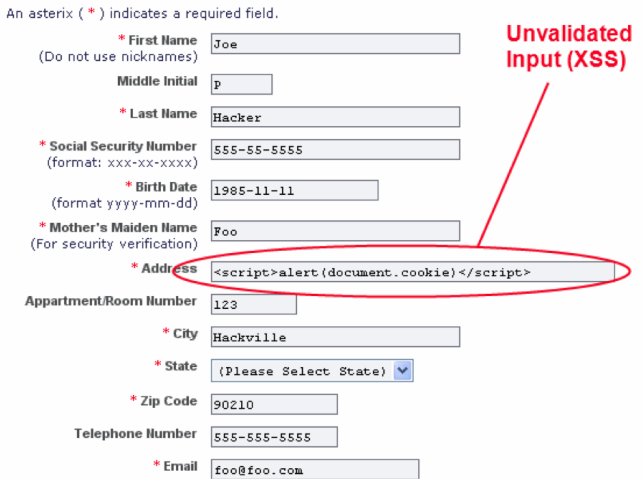
- Angreifer platziert seinen Schadcode auf einem Web-Server bei Zugriffen auf den Server durch Nutzer (Opfer) wird dieser Schadcode an den zugreifenden Nutzer weitergeleitet
- Schadcode i.d.R. in JavaScript, wird im Benutzer-Browser ausgeführt
- Web-Server sendet Nutzer-Eingaben in ungeprüfter bzw. ungefilterter Form an die Browser von Nutzern zurück.

Stored XSS (Type I)

Idee:

- Angreifer platziert seinen Schadcode auf einem Web-Server bei Zugriffen auf den Server durch Nutzer (Opfer) wird dieser Schadcode an den zugreifenden Nutzer weitergeleitet
- Schadcode i.d.R. in JavaScript, wird im Benutzer-Browser ausgeführt
- Web-Server sendet Nutzer-Eingaben in ungeprüfter bzw. ungefilterter Form an die Browser von Nutzern zurück.

Beispiel: Eintrag mit Scripttags in das Adressfeld einer Forumsregistrierung.



Reflected XSS (Type II)

Idee:

- Viele Eingabefelder (insbesondere Suchfelder) von Websites „spiegeln" die Eingabe an den Benutzer zurück
- Dem Opfer wird ein Link mit Einträgen auf ein derartiges Eingabefeld „untergeschoben". Sobald das Opfer diesen Link aufruft, wird der Browser des Opfers das gespiegelte Script ausführen.

Reflected XSS (Type II)

Idee:

- Viele Eingabefelder (insbesondere Suchfelder) von Websites „spiegeln“ die Eingabe an den Benutzer zurück
- Dem Opfer wird ein Link mit Einträgen auf ein derartiges Eingabefeld „untergeschoben“. Sobald das Opfer diesen Link aufruft, wird der Browser des Opfers das gespiegelte Script ausführen.

Beispiel:

1. Angreifer erzeugt eine URL, die ein Script enthält und sendet die URL an Opfer (z.B. E-Mail mit

```
http://www.amazon.de/field-keywords=<script>alert(1);</script>'
```

Reflected XSS (Type II)

Idee:

- Viele Eingabefelder (insbesondere Suchfelder) von Websites „spiegeln“ die Eingabe an den Benutzer zurück
- Dem Opfer wird ein Link mit Einträgen auf ein derartiges Eingabefeld „untergeschoben“. Sobald das Opfer diesen Link aufruft, wird der Browser des Opfers das gespiegelte Script ausführen.

Beispiel:

1. Angreifer erzeugt eine URL, die ein Script enthält und sendet die URL an Opfer (z.B. E-Mail mit `http://www.amazon.de/field-keywords=<script>alert(1);</script>'`
2. Opfer klickt auf URL

Reflected XSS (Type II)

Idee:

- Viele Eingabefelder (insbesondere Suchfelder) von Websites „spiegeln“ die Eingabe an den Benutzer zurück
- Dem Opfer wird ein Link mit Einträgen auf ein derartiges Eingabefeld „untergeschoben“. Sobald das Opfer diesen Link aufruft, wird der Browser des Opfers das gespiegelte Script ausführen.

Beispiel:

1. Angreifer erzeugt eine URL, die ein Script enthält und sendet die URL an Opfer (z.B. E-Mail mit `http://www.amazon.de/field-keywords=<script>alert(1);</script>'`
2. Opfer klickt auf URL
3. Script wird als Benutzer-Eingabe zum Server gesandt

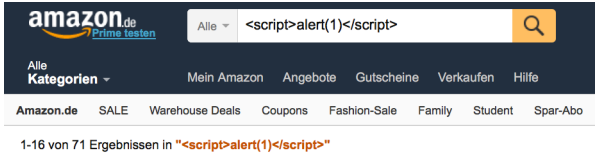
Reflected XSS (Type II)

Idee:

- Viele Eingabefelder (insbesondere Suchfelder) von Websites „spiegeln" die Eingabe an den Benutzer zurück
- Dem Opfer wird ein Link mit Einträgen auf ein derartiges Eingabefeld „untergeschoben". Sobald das Opfer diesen Link aufruft, wird der Browser des Opfers das gespiegelte Script ausführen.

Beispiel:

1. Angreifer erzeugt eine URL, die ein Script enthält und sendet die URL an Opfer (z.B. E-Mail mit `http://www.amazon.de/field-keywords=<script>alert(1);</script>'`
2. Opfer klickt auf URL
3. Script wird als Benutzer-Eingabe zum Server gesandt
4. Server spiegelt die Eingabe, also das Script, zurück an Nutzer



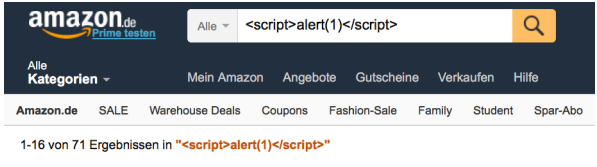
Reflected XSS (Type II)

Idee:

- Viele Eingabefelder (insbesondere Suchfelder) von Websites „spiegeln" die Eingabe an den Benutzer zurück
- Dem Opfer wird ein Link mit Einträgen auf ein derartiges Eingabefeld „untergeschoben". Sobald das Opfer diesen Link aufruft, wird der Browser des Opfers das gespiegelte Script ausführen.

Beispiel:

1. Angreifer erzeugt eine URL, die ein Script enthält und sendet die URL an Opfer (z.B. E-Mail mit `http://www.amazon.de/field-keywords=<script>alert(1);</script>'`
2. Opfer klickt auf URL
3. Script wird als Benutzer-Eingabe zum Server gesandt
4. Server spiegelt die Eingabe, also das Script, zurück an Nutzer



5. Script wird auf dem Client Rechner ausgeführt

DOM Injection XSS (Type 0)

Idee:

- Browser verarbeiten nicht nur Scripte, welche direkt vom Webserver geliefert werden, sondern interpretieren auch verschiedenste Objekte im Rahmen des Document-Object-Model *(DOM)*
- Enthält ein Objekt Script-Tags, welche der Browser nicht anders interpretieren kann, führt er diese u.U. aus

Beispiel:

1. Webseite enhält:

```
1 <select><script>
2 document.write("<OPTION value=1>"+document.location
      .href.substring(document.location.href.indexOf
      ("default=")+8)+"</OPTION>");
3 document.write("<OPTION value=2>English</OPTION>");
4 </script></select>
```

DOM Injection XSS (Type 0)

Idee:

- Browser verarbeiten nicht nur Scripte, welche direkt vom Webserver geliefert werden, sondern interpretieren auch verschiedenste Objekte im Rahmen des Document-Object-Model *(DOM)*
- Enthält ein Objekt Script-Tags, welche der Browser nicht anders interpretieren kann, führt er diese u.U. aus

Beispiel:

1. Webseite enhält:

```
1 <select><script>
2 document.write("<OPTION value=1>"+document.location
      .href.substring(document.location.href.indexOf
      ("default=")+8)+"</OPTION>");
3 document.write("<OPTION value=2>English</OPTION>");
4 </script></select>
```

2. User gibt ein:

```
http://www.some.site/page.html?default=<script>alert("Upsi")</script>
```

DOM Injection XSS (Type 0)

Idee:

- Browser verarbeiten nicht nur Scripte, welche direkt vom Webserver geliefert werden, sondern interpretieren auch verschiedenste Objekte im Rahmen des Document-Object-Model *(DOM)*
- Enthält ein Objekt Script-Tags, welche der Browser nicht anders interpretieren kann, führt er diese u.U. aus

Beispiel:

1. Webseite enhält:

```
1 <select><script>
2 document.write("<OPTION value=1>"+document.location
      .href.substring(document.location.href.indexOf
      ("default=")+8)+"</OPTION>");
3 document.write("<OPTION value=2>English</OPTION>");
4 </script></select>
```

2. User gibt ein:

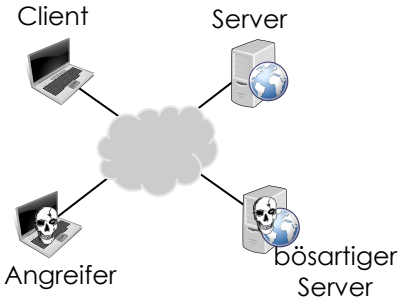
```
http://www.some.site/page.html?default=<script>alert("Upsi")</script>
```

3. Der Browser führt das ins Objekt eingebettete Script aus!

XSS Gegenmaßnahmen

Serverseite:

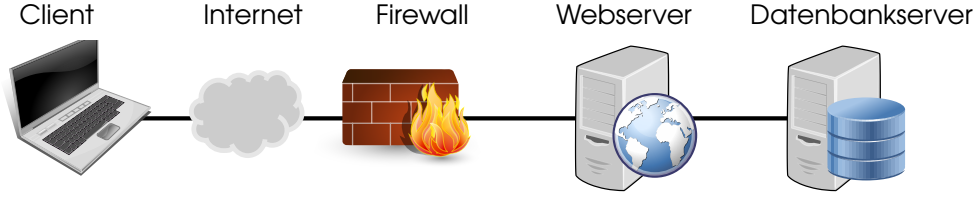
- Übergebene Parameter auf Metazeichen testen/filtern oder konvertieren
- Eingaben HTML-sicher konvertieren
- Beispiel:
  - Quotes of all kinds (‘, ”, and `)
  - Semicolons (;)
  - Asterisks (*)
  - Percents (%)
  - Underscores (_)
  - Other shell/scripting metacharacters (=&\—*? <>ˆ ()[]{}$\n \r)



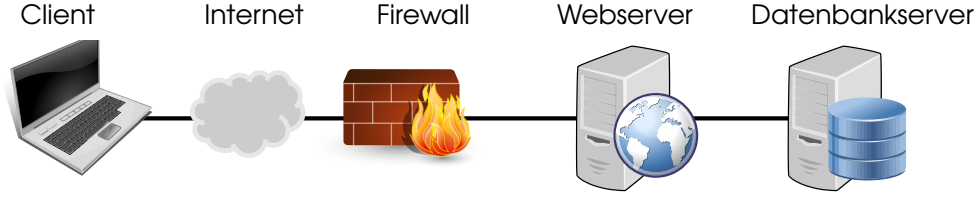
Clientseite:

- Scripte nur von „wirklich“ vertrauenswürdigen Seiten erlauben
- sichere Browser oder Browserplugins verwenden
- Vorsicht mit Links aus unverifizierten Quellen

SQL Injection

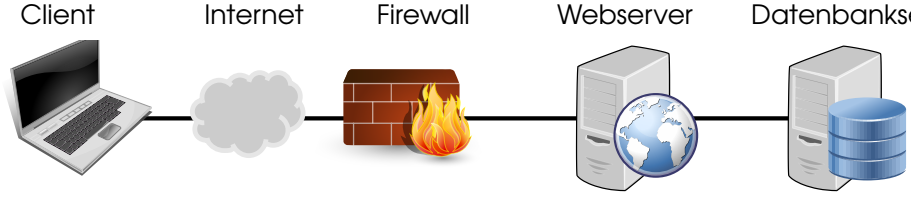


SQL Injection



Eingabe von Daten in ein Webformular (z.B. Name)

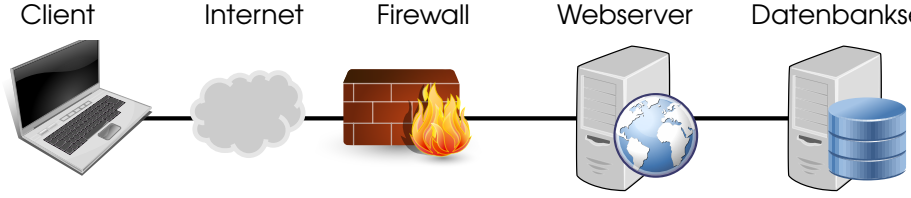
SQL Injection



Eingabe von Daten in ein Webformular (z.B. Name)

Erstellen eines SQL-Befehls aus der Benutzereingabe

SQL Injection

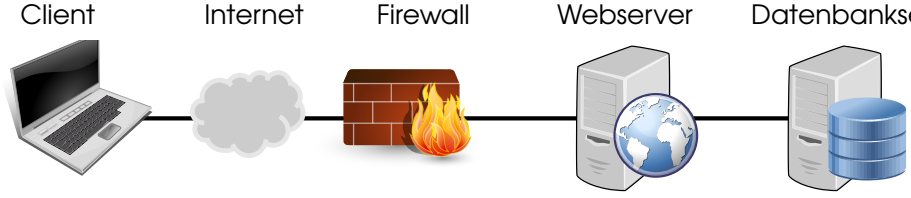


Eingabe von Daten in ein Webformular (z.B. Name)

Erstellen eines SQL-Befehls aus der Benutzereingabe

Senden des Befehls an die Datenbank

SQL Injection



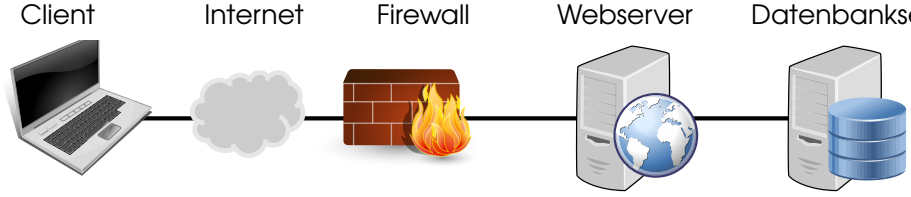
Eingabe von Daten in ein Webformular (z.B. Name)

Erstellen eines SQL-Befehls aus der Benutzereingabe

Senden des Befehls an die Datenbank

Verarbeitung des Befehls und Antwort an den Webserver

SQL Injection



Eingabe von Daten in ein Webformular (z.B. Name)

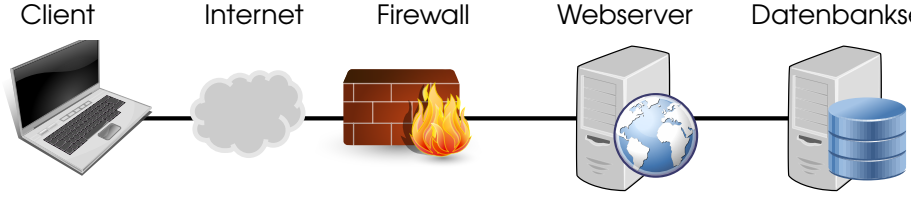
Erstellen eines SQL-Befehls aus der Benutzereingabe

Senden des Befehls an die Datenbank

Verarbeitung der Antwort, Erstellung der angepassten Webseite für den Client

Verarbeitung des Befehls und Antwort an den Webserver

SQL Injection



Eingabe von Daten in ein Webformular (z.B. Name)

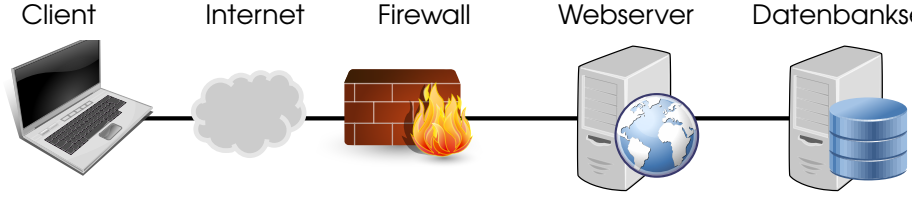
Erstellen eines SQL-Befehls aus der Benutzereingabe

Senden des Befehls an die Datenbank

Verarbeitung der Antwort, Erstellung der angepassten Webseite für den Client

Verarbeitung des Befehls und Antwort an den Webserver

SQL Injection



Eingabe von Daten in ein Webformular (z.B. Name)

Erstellen eines SQL-Befehls aus der Benutzereingabe

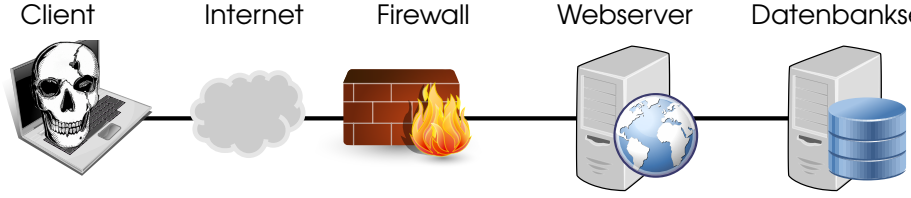
Senden des Befehls an die Datenbank

Verarbeitung des Befehls und Antwort an den Webserver

Verarbeitung der Antwort, Erstellung der angepassten Webseite für den Client

Anzeige der angepassten Webseite

SQL Injection



Eingabe von manipulierten Daten in ein Webformular (z.B. SQL-Kommandos ins Namensfeld)

Erstellen eines SQL-Befehls aus der Benutzereingabe

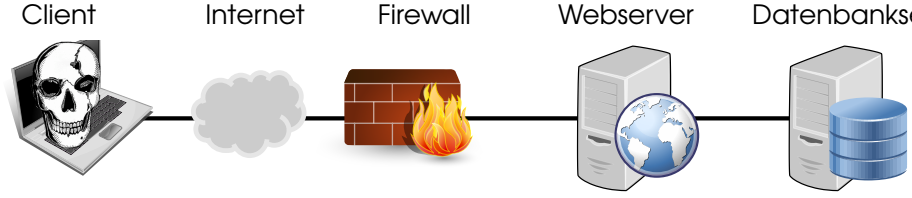
Senden des Befehls an die Datenbank

Verarbeitung des Befehls und Antwort an den Webserver

Verarbeitung der Antwort, Erstellung der angepassten Webseite für den Client

Anzeige der angepassten Webseite

SQL Injection



Eingabe von manipulierten Daten in ein Webformular (z.B. SQL-Kommandos ins Namensfeld)

Erstellen eines SQL-Befehls aus der manipulierten Benutzereingabe

Senden des Befehls an die Datenbank

Verarbeitung der Antwort, Erstellung der angepassten Webseite für den Client

Verarbeitung des Befehls und Antwort an den Webserver

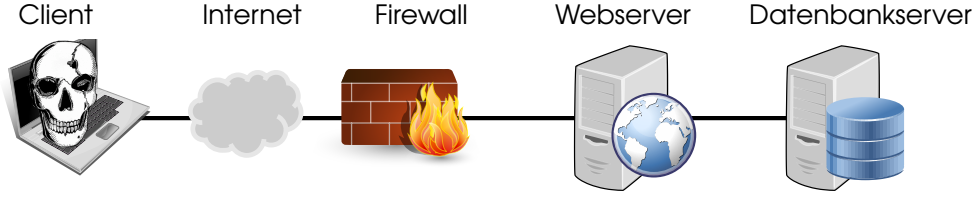
Anzeige der angepassten Webseite

SQL Injection Beispiel 1: User-Passwort-Check

Häufige php-Aufgabe:

Überprüfung der Übereinstimmung einer Eingabe mit einem Eintrag in der Datenbank

(z.B. Username, Passwort-Überprüfung)



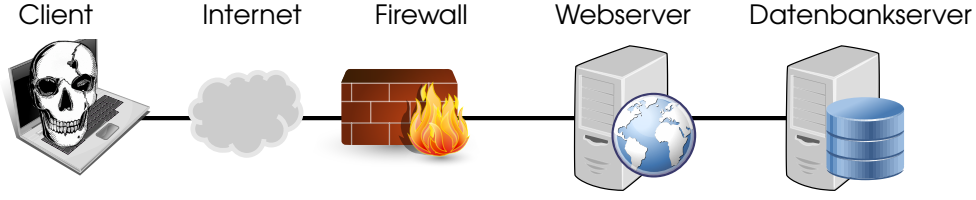
Eingabe ins Passwortfeld: `none' or 1=1 or password='root`

SQL Injection Beispiel 1: User-Passwort-Check

Häufige php-Aufgabe:

Überprüfung der Übereinstimmung einer Eingabe mit einem Eintrag in der Datenbank

(z.B. Username, Passwort-Überprüfung)



Eingabe ins Passwortfeld: `none' or 1=1 or password='root`

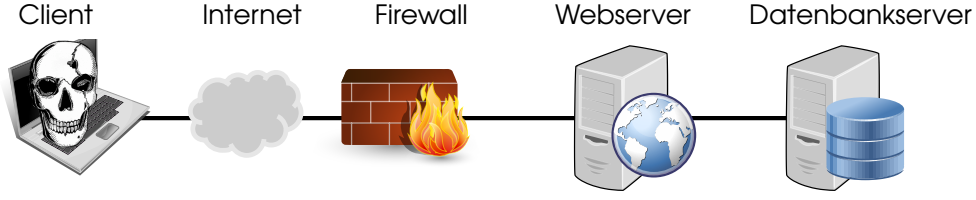
Wird zu:
```
mysql_query(select user_id from users where username='$user'
                and password='none' or 1=1 or password='root');
```

SQL Injection Beispiel 1: User-Passwort-Check

Häufige php-Aufgabe:

Überprüfung der Übereinstimmung einer Eingabe mit einem Eintrag in der Datenbank

(z.B. Username, Passwort-Überprüfung)



Eingabe ins Passwortfeld: `none' or 1=1 or password='root`

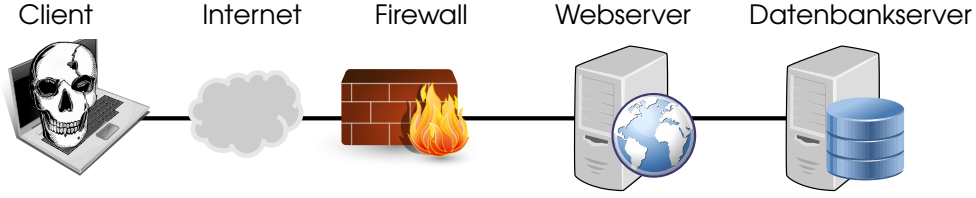
Wird zu:
```
mysql_query(select user_id from users where username='$user'
                and password='none' or 1=1 or password='root');
```

Rückgabe sämtlicher `user_id`

SQL Injection Beispiel 2: Befehl einschleusen

noch besser:

Einschleusen eines völlig eigenen SQL-Befehls

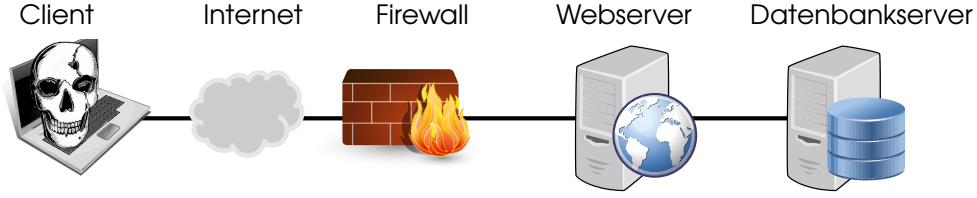


Eingabe in ein Namensfeld: `max'; SELECT salary FROM personal WHERE name='fritz`

SQL Injection Beispiel 2: Befehl einschleusen

noch besser:

Einschleusen eines völlig eigenen SQL-Befehls



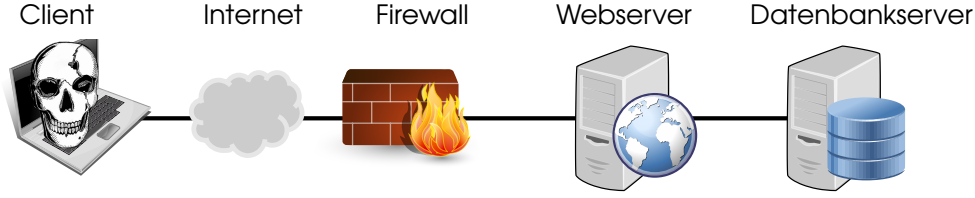
Eingabe in ein Namensfeld: `max'; SELECT salary FROM personal WHERE name='fritz`

Wird zu:
```
SELECT age FROM user WHERE name = 'max';
SELECT salary FROM personal WHERE name='fritz';
```

SQL Injection Beispiel 2: Befehl einschleusen

noch besser:

Einschleusen eines völlig eigenen SQL-Befehls



Eingabe in ein Namensfeld: `max'; SELECT salary FROM personal WHERE name='fritz`

Wird zu:
```
SELECT age FROM user WHERE name = 'max';
SELECT salary FROM personal WHERE name='fritz';
```

Rückgabe des Gehalts von `fritz`

Fazit SQL-Injection

- Datenbankserver über Web angreifbar
  - Rechte für Datenbankbenutzer einschränken; need-to-know
  - Datenbankabfragen nicht aus Strings zusammensetzen.
  - Nutzung von APIs mit separierten Aufrufen für Struktur und Inhalt

    z.B. In PHP `mysql_real_escape_string()` verwenden
- Ursache vieler Web-App-Schwachstellen:
  - fehlende Eingabe-Prüfung/Eingabe-Filterung/Input Validation
  - Schwachstellen ergeben sich aus Programmierfehlern!
- Benötigt: einmal mehr!
  - Regeln zum sicheren Programmierung, Tool-Unterstützung!
  - Eingabe-Filterung/Escaping etc.

Danke für Ihr Interesse

Fortsetzung folgt