

# Web-Security

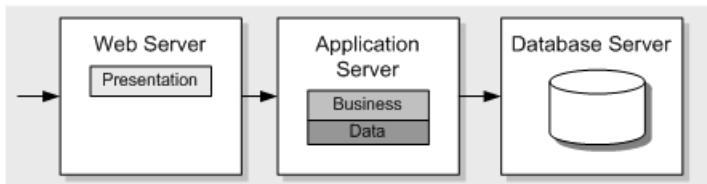
## IT-Security

Andreas Unterweger

Studiengang Web Business & Technology  
FH Kufstein

Sommersemester 2020

- Webseitensicherheit
  - (Kurz-)Wiederholung JavaScript
  - Cross-Site Scripting
  - Cross-Site Request Forgery
- Datenbanksicherheit
  - (Kurz-)Wiederholung SQL
  - SQL Injection



Quelle: Meier, J. D., Homer, A., Hill, D., Taylor, J., Bansode, P., Wall, L., Boucher, R. und Bogawat, A.: Chapter 5: Deployment Patterns. <http://apparchguide.codeplex.com/wikipage?title=Chapter%205%20-%20Deployment%20Patterns> (Zugriff am 16.4.2017), 2008.

- Beispiel für AJAX per XMLHttpRequest:

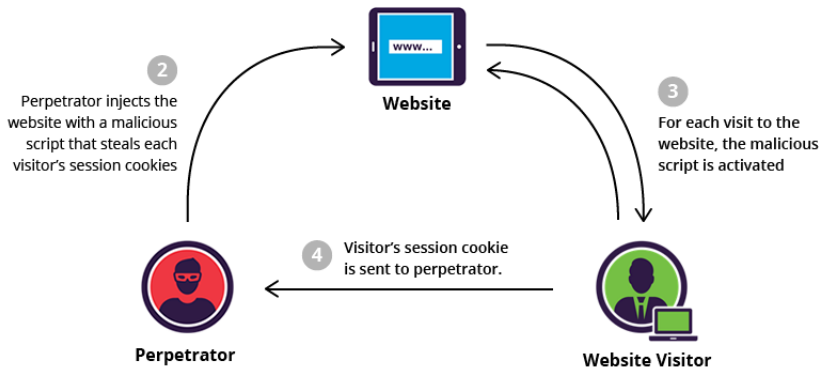
```
1 var xhttp = new XMLHttpRequest();
2 xhttp.onreadystatechange = function() {
3     if (this.readyState == 4 && this.status == 200)
4     {
5         // Typical action to be performed when the
           document is ready:
6         document.getElementById("demo").innerHTML =
           xhttp.responseText;
7     }
8 };
9 xhttp.open("GET", "filename", true);
10 xhttp.send();
```

Adaptiert von W3Schools: XML HttpRequest. [https://www.w3schools.com/xml/xml\\_http.asp](https://www.w3schools.com/xml/xml_http.asp) (Abruf am 16.4.2017), 2017.

# Überblick Cross-Site Scripting (XSS)

- Webseite verarbeitet Benutzereingabe, z.B. Forenbeitrag
- Ungefilterte Eingabe (z.B. Skript) erlaubt Angriffe bei erneutem Aufruf/Besuch der Webseite durch Opfer
  - Bei Verwendung von HTML: HTML Injection
  - Bei Verwendung von JavaScript: JavaScript Injection
- Arten von XSS (Auswahl)
  - Nicht-persistent (reflektiert)
  - Persistent (beständig)
  - DOM-basiert
- Mögliche Ziele eines Angreifers (Auswahl)
  - Daten von oder über Opfer(-rechner) sammeln
  - Dargestellte Webseite manipulieren (Täuschung)
  - Opfer auf andere Webseiten umleiten
  - Schadcode ausführen

# Beispiel für Cross-Site Scripting

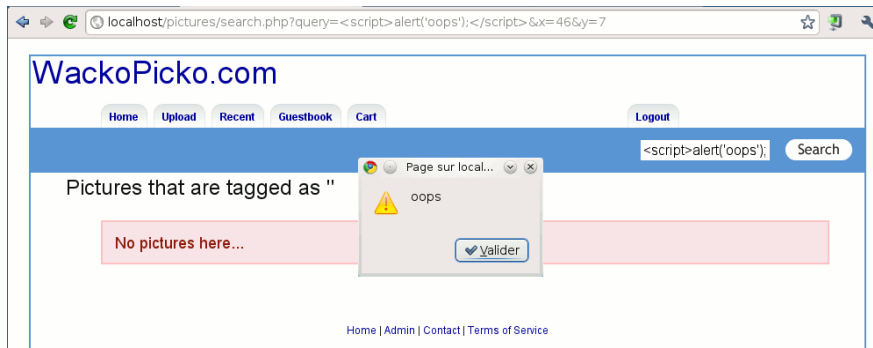


Quelle: Imperva: Cross Site Scripting (XSS) Attacks.

<https://www.incapsula.com/web-application-security/cross-site-scripting-xss-attacks.html> (Zugriff am 16.4.2017), 2017.

# Nicht-persistentes XSS

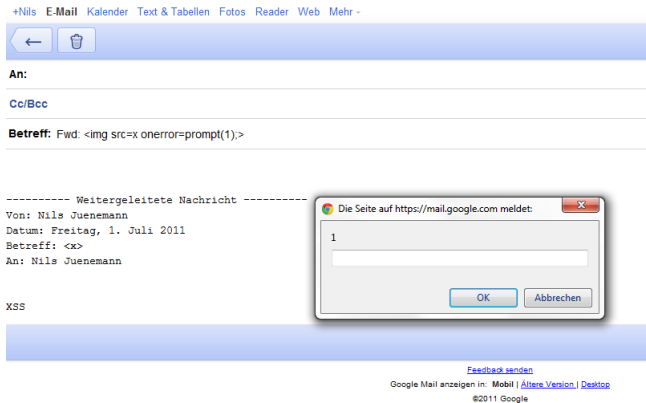
- Benutzereingabe wird reflektiert, d.h. wieder ausgegeben
- Beispiel: Suchbegriff (reflektiert in Ergebnisliste) – am Server: Pictures that are tagged as '<?=\$\_GET['query'] ?>'



Quelle: Damaye, S.: WackoPicko/Reflected-XSS. <https://www.aldeid.com/wiki/WackoPicko/Reflected-XSS> (Zugriff am 16.4.2017), 2013.

# Persistentes XSS

- Benutzereingabe wird dauerhaft gespeichert, z.B. als Seiteninhalt
- Beispiel: Im Browser dargestellte Email mit eingebettetem Bild



The screenshot shows a Gmail interface with a forwarded email. The email header includes: +Nils E-Mail Kalender Text & Tabellen Fotos Reader Web Mehr -; An:; Cc/Bcc; and Betreff: Fwd: <img src=x onerror=prompt(1)>. The email body contains a forwarded message from Nils Juenemann dated Friday, July 1, 2011, with the subject <x> and recipient Nils Juenemann. Below the email content, the text 'XSS' is visible. An error dialog box is overlaid on the screen, titled 'Die Seite auf https://mail.google.com meldet:' with a close button (X). The dialog contains the number '1' and a text input field. At the bottom of the dialog are 'OK' and 'Abbrechen' buttons. Below the email content, there is a 'Feedback senden' link and footer text: 'Google Mail anzeigen in: Mobil | [Ändere Version](#) | [Desktop](#) ©2011 Google'.

Quelle: Lev, S.: Cross-Site-Scripting in Google Mail.

<https://0xicf.wordpress.com/2012/06/13/cross-site-scripting-in-google-mail/> (Zugriff am 16.4.2017), 2012.

- Ausgangspunkt: Clientseitige Skripte verändern Webseite (legitimerweise) über das DOM (besonders über AJAX)
- Veränderungen sind von Benutzereingaben abhängig
- Unüberprüfte Eingaben erlauben bösartige Veränderungen
- Server ist **nicht** involviert
- Anfällige Funktionalität (Auswahl):
  - `<DOM-Element>.innerHTML`: Ersetzt den Code des DOM-Elements
  - `document.write`: Ergänzt das HTML-Dokument
  - `document.location.href`: Lädt eine (andere) Seite
- Beispiel: (Dynamische) Sprachauswahl mit Standardwert (Parameter):  
`http://www.spielwiese.tld/index.html?lang=Deutsch`



# DOM-basiertes XSS II

```
1 <!-- ... -->
2 Anzeigesprache:
3 <select>
4   <script>
5     document.write("<option value=1>" + document.
6       location.href.substring(document.location.
7       href.indexOf("lang=") + 5) + "</option>");
8   <!-- Weitere Sprachen... -->
9 </script>
10 </select>
11 <!-- ... -->
```

- Bösartiger Aufruf (z.B. als Link per Email):

```
http://www.spielwiese.tld/index.html?lang=<script>
  document.querySelector('body > :nth-child(1)').
  innerHTML="Schadcode..."</script>
```

Adaptiert von OWASP Contributors: DOM Based XSS. [https://www.owasp.org/index.php/DOM\\_Based\\_XSS](https://www.owasp.org/index.php/DOM_Based_XSS) (Zugriff am 16.4.2017), 2015.

# Vermeidung von Cross-Site Scripting

- Benutzereingaben überprüfen (mehrere Möglichkeiten)
  - Kritische Zeichen (z.B. <) verbieten
  - Kritische Zeichen escapen (z.B. < durch &lt; ersetzen)

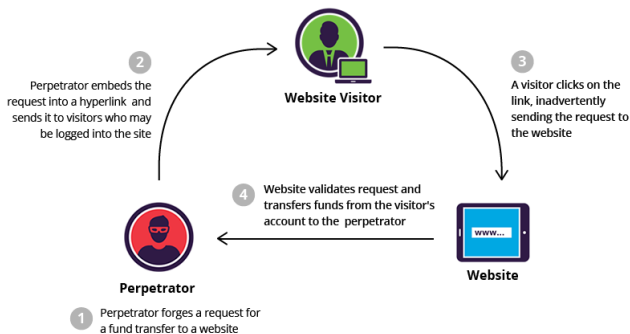
→ Funktionen (z.B. `htmlspecialchars` in PHP) zum Escapen

Kritisches Zeichen	Ersetzung	Anmerkung
&	<code>&amp;amp;</code>	
<	<code>&amp;lt;</code>	
>	<code>&amp;gt;</code>	
"	<code>&amp;quot;</code>	
'	<code>&amp;#x27;</code>	<code>&amp;#x39;</code> in <code>htmlspecialchars</code>
/	<code>&amp;#x2F;</code>	In <code>htmlspecialchars</code> nicht ersetzt

- Weitere Empfehlungen: [https://www.owasp.org/index.php/XSS\\_\(Cross\\_Site\\_Scripting\)\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)

# Cross-Site Request Forgery (CSRF)

- Angreifer bringt Opfer dazu, Anfrage (engl. *request*) auszuführen
- Wirkt, als würde Anfrage (z.B. Überweisung) von Opfer kommen



Quelle: Imperva: Cross Site Request Forgery (CSRF) Attack.

<https://www.incapsula.com/web-application-security/csrf-cross-site-request-forgery.html> (Zugriff am 16.4.2017), 2017.

- Referrer-Überprüfung
  - Referrer: Feld in HTTP-Header, das den Ursprung der Anfrage angibt
  - Anfragen nur von bestimmten Quellen erlauben
  - Lässt sich relativ leicht umgehen (ohne Details)
- Anti-CSRF-Tokens
  - Webseite generiert (einmaliges) Token bei Aufruf der Webseite
  - Formularen auf der Webseite enthalten (verstecktes) Token
  - Abgeschickte Formulardaten werden nur mit gültigem Token verarbeitet
  - Token ist dem Angreifer nicht bekannt → keine Verarbeitung
  - **Aber:** Tokens lassen sich per XSS auslesen und dadurch umgehen
- Moderneres Beispiel: *Access Control for Cross-Site Requests*<sup>1</sup>
  - Browser überprüft, ob Antworten auf Anfragen von der gleichen Seite (oder einer erlaubten anderen) kommen (*Same Origin Policy*)
  - Schwieriger zu umgehen (ohne Details)

---

<sup>1</sup><https://www.w3.org/TR/2014/REC-cors-20140116/>

- Beispiel für die Erstellung und Befüllung einer Datenbank(-tabelle):

```
1 CREATE DATABASE webshop;
2 USE webshop;
3 CREATE TABLE products (
4     id INT AUTO_INCREMENT PRIMARY KEY,
5     name VARCHAR(20) NOT NULL,
6     price DECIMAL(4, 2) NOT NULL
7 );
8 INSERT INTO products (name, price) VALUES ("T-
9     Shirt", "0019.90");
9 -- Weitere Befehle... --
```

- Beispiel für eine Abfrage:

```
SELECT * FROM products WHERE name="T-Shirt";
```

Basierend auf dem MySQL 5.5 Reference Manual (<https://dev.mysql.com/doc/refman/5.5/en/>)

- Datenbankabfrage hängt von Benutzereingabe, z.B. Suchbegriff, ab
- Ungefilterte Eingabe erlaubt Angriffe über Datenbank(-abfrage)
- Arten von SQL Injection (Auswahl)
  - **UNION-basiert**: Zusätzliche Datenbankabfrage ausführen und mit Ergebnis der ursprünglichen Abfrage zusammenführen
  - Blind: Informationen über Antwortzeit(en) rekonstruieren
- Mögliche Ziele eines Angreifers (Auswahl)
  - Erweiterte Suchabfragen absetzen
  - Zusätzliche Datenbanktabellen abfragen → Informationsbeschaffung
  - Datenbanksystem und -version auslesen (*Banner Grabbing*)
  - Daten in Datenbank manipulieren oder löschen (falls unterstützt)
  - Dateien auf dem Datenbankserver auslesen (falls unterstützt)

# Funktionsweise von SQL Injections I: Beispiel

- Ursprüngliche Datenbankabfrage mit Eingabeparameter \$name

```
SELECT * FROM products WHERE name = "$name";
```

- Speziell konstruierter Parameter \$name:

```
" OR name = "secret"; --
```

- Datenbankabfrage mit speziell konstruiertem Parameter \$name:

```
SELECT * FROM products WHERE name = " " OR name = "secret";  
--";
```

- Effektiv ausgeführte Datenbankabfrage:

```
SELECT * FROM products WHERE name = " " OR name = "secret";
```

```
"_OR_name="secret";_--_"
```

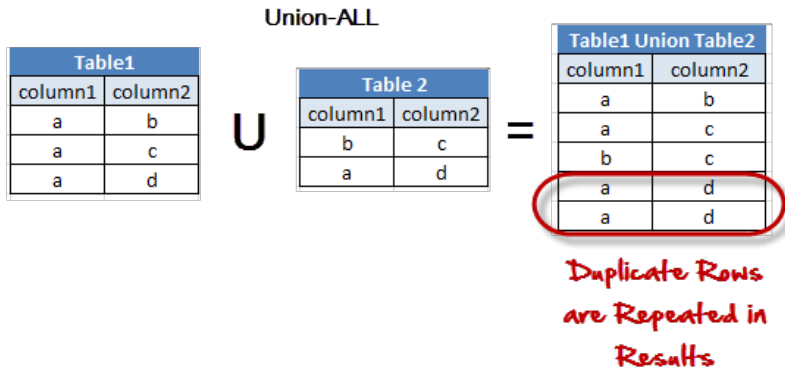
- Anatomie des speziell konstruierten Parameters:
  - Anführungszeichen, um jene vor dem Eingabeparameter zu schließen
  - Neue Abfrage in SQL-Syntax (hängt von ursprünglicher Abfrage ab)
  - Kommandoendesymbol (;), um Abfrage abzuschließen
  - Kommentarsymbol (--), um restliche Zeichen der Abfrage auszukommentieren (vermeidet u.a. weitere Filterungsbefehle)
  - **Leerzeichen** nach dem Kommentarsymbol (in SQL notwendig)
- Konstruktion vermeidet Syntaxfehler
- Art und Funktion von neuer Abfrage hängt vom Aufbau der ursprünglichen Datenbankabfrage (Befehle vor dem Parameter) ab
- Praktische Variation: Alle Daten auslesen, z.B. über Tautologie:

```
"_OR_'x'='x';_--_"
```



# Einschub: UNION ALL in SQL

- Vereinigungsmenge zweier Tabellen oder Abfragen **mit** Duplikaten



Quelle: Guru99: MySQL UNION - Complete Tutorial. <http://www.guru99.com/unions.html> (Zugriff am 15.4.2017), 2017.

# UNION-basierte SQL Injection I: Beispiel

- Idee: Zusätzliches SELECT-Statement ausführen und Ergebnis per UNION mit dem Originalstatement zusammenführen
- Ausgangspunkt (wie vorhin):

```
SELECT * FROM products WHERE name = "$name";
```

- Speziell konstruierter Parameter \$name:

```
" UNION ALL SELECT name FROM customers; --
```

- Effektiv ausgeführte Datenbankabfrage:

```
SELECT * FROM products WHERE name = "" UNION ALL SELECT name FROM customers;
```

```
" UNION ALL SELECT name FROM customers ; -- "
```

- Anatomie des speziell konstruierten Parameters:
  - Anführungszeichen, um jene vor dem Eingabeparameter zu schließen
  - UNION ALL, um Vereinigungsmenge der beiden Ergebnisse zu bilden
  - Neue Abfrage (hängt **nicht** von ursprünglicher Abfrage ab)
  - Abschluss (vgl. Folie 16)
- Originalabfrage ist leer (oder hat nur sehr wenige Einträge)
- Ergebnis der zusätzlichen Abfrage wird zu Originalergebnis ergänzt<sup>2</sup>
- Voraussetzung für erfolgreiches Vereinigen: Abfragen haben **exakt die gleiche** Spaltenanzahl

---

<sup>2</sup>Falls unerwünscht: AND 1=0 (oder andere falsche Aussage) hinzufügen, um leeres Ergebnis für Originalabfrage zu erzwingen

- Spaltenanzahl per SQL Injection ermitteln
- Idee: Sortierung per ORDER BY mit Spalten**nummer** angeben:

```
" ORDER BY 3; --"
```

- Falls Spaltennummer nicht vorhanden → Fehler
  - Falls Spaltennummer vorhanden → Spaltenzahl ist größer oder gleich
- Spaltenanzahl durch Ausprobieren ermitteln
- Bei bekannter Anzahl von Spalten (z.B. 3) SQL Injection mit Auswahl aller Spalten in SELECT-Statement:

```
" UNION ALL SELECT 1, 2, 3 FROM products; --"
```

- Befehle zur Abfrage von Datenbanksysteminformationen (Auswahl):
  - `version()`: Datenbankversion
  - `user()`: Benutzer, in dessen Kontext die Datenbank läuft
- Beispiel mit SQL Injection:

```
" UNION ALL SELECT 1, user(), 3 FROM products; --"
```

- Befehle zur Abfrage von Datenbankinformationen (Auswahl):
  - `database()`: Name der Datenbank
  - Datenbanksystemabhängige Funktionen
- Befehle zum Lesen von Systeminformationen (Auswahl):
  - `load_file(path)`: Liest den Inhalt der Datei `pfad`
  - ...
- **Spezielle Datenbanktabellen mit relevanten Informationen**

# Einschub: Spezielle Datenbanktabellen in SQL

- Datenbank INFORMATION\_SCHEMA enthält Informationen über alle (anderen) Datenbanken
  - Tabelle SCHEMATA: Liste von Datenbanken
  - Tabelle TABLES: Liste von Datenbanktabellen
  - Tabelle COLUMNS: Liste von Datenbanktabellenspalten

```
1 select * from INFORMATION_SCHEMA.COLUMNS
2
```

121 %

Results Messages

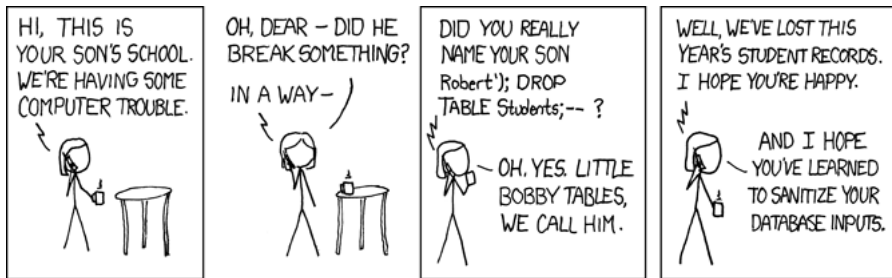
	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	DATA_TYPE
1	Training	dbo	Temp_emp_table	id	1	int
2	Training	dbo	Temp_emp_table	name	2	varchar
3	Training	dbo	Temp_emp_table	mobile	3	int
4	Training	dbo	Customer_s	CustID	1	int
5	Training	dbo	Customer_s	FName	2	varchar
6	Training	dbo	Customer_s	LName	3	varchar
7	Training	dbo	Customer_s	UserID	4	varchar

Adaptiert von Choudhary, P. K.: Get Columns Related Information In SQL Server.

<http://www.c-sharpcorner.com/blogs/difference-bw-informationschemacolumns-and-syscolumns> (Zugriff am 15.4.2017), 2016.

# Vermeidung von SQL Injections

- Benutzereingaben überprüfen und „entschärfen“, z.B. Anführungszeichen escapen (\" statt ")
- In der Praxis: Serverseitige Bibliotheksfunktionen für Escaping
- **Zusätzlich:** Rechteinschränkung für Datenbankbenutzer
    - Leserechte nur auf unbedingt benötigte Datenbanken bzw. Tabellen
    - Keine oder nur minimale Modifikationsmöglichkeiten



Quelle: Munroe, R.: Exploits of a Mom. <https://xkcd.com/327/> (Zugriff am 15.4.2017), unbekannt.

Fragen?