

Letzte Vorlesung

Themen der letzten Vorlesung:

- Sicherheit in Internet-Anwendungen
- Buffer-Overflow-Attacken
- Code einschleusen: Shellcode

Schlechten Ideen (1)

Sensible Daten im "versteckten" Feldern transportieren:

```
<input type="hidden" name="price" value="344,95" />
```

Möglichkeiten zur Manipulation:

- Bei Übertragung per **GET**: URL im Browser editieren
- Web-Seite speichern und editieren
- Fingierten **POST**-Request absetzen

Schlechte Ideen (2)

Sensible Daten per **GET** übergeben

```
http://.../shop.cgi?do=add&productid=234&price=344.95
```

Möglichkeiten zur Manipulation:

- Bei Übertragung per **GET**: URL im Browser editieren
- Fingierten **POST**-Request absetzen

Manipulation

Mantra:

Alle Daten, die vom Client kommen, können manipuliert sein

Das heißt:

- Versteckte Eingabefelder
- Per **GET** oder **POST** übergebene Daten
- Daten aus Cookies

⇒ **Sensible Daten nicht an Client weitergeben. Dies gilt insbesondere für Session-Daten.**

Bessere Lösung:

Lediglich Referenz auf Daten an Client weitergeben

Sichere Session-Daten

Sichere Speicherung von Session-Daten:

- Identifiziert durch eindeutige, lange, nicht zu ratende Session-ID
- Session-Daten serverseitig speichern
- Session-Daten mit Verfallsdatum versehen
- Geschlossene Sessions nicht wiedererwecken
- Dem Benutzer die Möglichkeit geben, die Session explizit zu schließen

Diese Empfehlungen bieten *keine absolute* Sicherheit!

Beispiel für so eine Session-Verwaltung: PHP

Cross-Site-Scripting

Cross-Site-Scripting (XSS)

Browser führen den JavaScript-Code einer (vertrauenswürdigen) Web-Seite aus.

Gelingt es fremden JavaScript-Code einzuschmuggeln, wird dieser ebenfalls ausgeführt.

Wo könnte Code eingeschleust werden?

- Webmail-Anwendung, die eine HTML-Mail anzeigt
- Beiträge in einem Web-Forum
- Kommentarfunktionen
- Alle Anwendungen, die Eingaben vom Benutzer darstellen

Einfacher Fall

Einfachstes Beispiel: Webmail-Software, die eine HTML-Mail darstellt.

Einzuschmuggelnder Code: Innerhalb eines `script`-Element im Rumpf der E-Mail

Der Browser unterscheidet nicht zwischen JavaScript-Code, der zur E-Mail gehört und Code, der zur Webmail-Anwendung gehört.

⇒ JavaScript-Code wird ausgeführt

Konsequenzen

Was z. B. kann eingeschmuggelte JavaScript-Code anrichten?

- Die Cookie-Daten per Request an einen Dritten übermitteln
- Sensible Informationen der Web-Seite auslesen und ändern
- Zugriff auf geschützte, aber für den Client zugängliche Web-Seiten bekommen
- ...

Gegenmaßnahmen

Web-Applikationen müssen Gegenmaßnahmen ergreifen:

- Eingaben nie ungeprüft auf einer Seite ausgeben
- Bei Eingabe von HTML-Code: Nur bestimmte Tags erlauben

Vorsicht: Häufig wird versucht das `script`-Tag zu "verschleiern":

- `<SCRIPT a="" SRC="http://.../xss.js"></SCRIPT>`
- `<SCRIPT>document.write("<SCRI");</SCRIPT>
PT SRC="http://.../xss.js"></SCRIPT>`
- `<script>alert("XSS");<script>`

⇒ Überprüfung anhand regulären Ausdrucks schwierig

JavaScript überall

Web-Browser führen allerdings nicht nur den JavaScript-Code innerhalb von `script`-Elementen aus:

- ``
- ``
- ``

⇒ Es gibt viele Möglichkeiten JavaScript-Code so zu formulieren, dass eine Applikation diesen trotz Überprüfung nicht bemerkt.

Gegenmaßnahmen (2)

Die Überprüfung von eingegebenen Daten, die später dargestellt werden, sollte

- sehr streng sein
- verschiedene Kodierungen der Eingabe berücksichtigen
- sofern die Eingabe kein HTML enthält, alle HTML-Spezialzeichen quoten:
`<script>` anstatt `<script>`

SQL Injection

In der Praxis verwenden Web-Applikationen häufig Datenbanken, um Daten zu speichern.

Beispiel:

- Login-Formular mit Username und Passwort
 - Username und Passwort sind in Datenbank gespeichert
- ⇒ Um das Passwort zu überprüfen, muss eine Datenbankabfrage durchgeführt werden.

Structured Query Language

Verbreitete Sprache für Datenbankabfragen:

Structured Query Language (SQL)

- Anfänge gehen zurück auf die 70er Jahre
- Es gibt einen Standard. Einzelne DB-Produkte haben häufig ihren eigenen Dialekt
- Daten werden in Tabellen repräsentiert
- SQL beschreibt Anfragen und Operationen auf diesen Tabellen

SQL: Suche

Nach bestimmten Einträgen einer Tabelle suchen:

```
SELECT spalte-1, ... FROM tabellen-name  
WHERE bedingung
```

Beispiel:

```
SELECT userid, password FROM accounts  
WHERE userid='eric@dev.null'
```

- Gibt eine zweispaltige Tabelle zurück: `userid`, `password`
- Alle Einträge aus `accounts` enthält, deren `userid` `eric@dev.null` entspricht

Ablauf Web-Applikation

Zur Überprüfung des Passworts in der Web-Applikation:

- Entgegennehmen der Eingaben aus dem Formular
- Mit Datenbank verbinden
- Anfrage (als String) zusammenstellen und absetzen
- Ergebnis empfangen
- Eingegebenes Passwort mit Passwort aus der DB vergleichen

Interessant: Anfrage zusammenstellen

Anfrage zusammenstellen

Zusammenstellen der SQL-Anfrage:

```
$query = "SELECT userid, password FROM accounts WHERE ".  
        "userid='".$_REQUEST['userid']."'";
```

Jetzt sollten die Alarmglocken schrillen! Weshalb?

Benutzer-Eingabe wird ungeprüft weiterarbeitet.

Was kann passieren?

SQL Injection

SQL Injection

Gezielt Strings einfügen, die die Semantik der SQL-Anfrage ändern.

Beispiel: `eric@dev.null'`

⇒ Löst einen SQL-Syntax-Fehler aus

Beispiel: `egal' OR '23'='23'`

Führt zu:

```
SELECT userid, password FROM accounts  
WHERE userid='egal' OR '23'='23'
```

⇒ Ergebnis enthält komplette Tabelle, Reaktion der Applikation unklar

Beispiele für SQL-Injection

Drastischeres Beispiel für SQL-Injection:

```
egal'; DROP TABLE accounts;
```

```
SELECT userid, password FROM accounts  
WHERE userid='egal'; DROP TABLE accounts; OR '23'='23'
```

⇒ Löscht die komplette Tabelle

Ähnlich durchführbar: Tabelleneinträge hinzufügen

Gegenmaßnahmen

Gegenmaßnahmen SQL Injection:

- Zeichen in der Eingabe mit besonderer Bedeutung für SQL entsprechend quoten
- Nutzungsrechte für DB aufteilen und minimieren

In der Praxis: Namen der Tabelle und Felder nicht bekannt

- Es gibt Mittel die Namen herauszufinden
- Fehlermeldungen, die Namen der Tabelle und Felder enthalten nur lokal ausgeben