IT-Sicherheit

Sicherheit von Web-Anwendungen (Teil 1)

Prof. Dr. Dominik Merli, Prof. Dr. Lothar Braun

Sommersemester 2020

Hochschule Augsburg - Fakultät für Informatik

Web-Anwendungen

Was ist eine Web-Anwendung?

- Anwendung nach dem Client/Server Modell
  - Client: Webbrowser
  - Server: Webserver
- Kommunikationsprotokoll
  - HTTP (unverschlüsselt)
  - HTTPS (HTTP geschützt durch TLS-Kanal)
- Typische Programmiersprachen
  - HTML, PHP, Javascript
  - Ruby, Python, Go
  - Java, etc.

Welche grundlegende Eigenschaft
des Client/Server Modells ist
äußerst relevant für die IT-Sicherheit?

Einfache Web-Anwendung

- Ablauf einer HTTP-Verbindungen
  - Client öffnet TCP-Verbindung zu Web-Server und schickt HTTP-Anfrage
  - Server bearbeitet Request und schickt HTTP-Antwort
- Protokoll hält keinen Zustand
  - Jeder Request wird unabhängig von anderen Anfragen bearbeitet
  - Zustand muss durch Web-Anwendung modelliert werden

HTTP Grundlagen

- Applikationsprotokoll
  - HTTP/1.1 → RFC 723X
  - HTTP/2 → RFC 7540
- Datenaustausch zwischen Client und Server
  - Durch verschiedene Anfragen (engl. requests)
  - z.B. GET, POST, HEAD, PUT, DELETE, etc.
- Requests bestehen aus Header und Body
- Kommunikation auf Ports 80 (HTTP) und 443 (HTTPS)

HTTP Statuscodes

- 1xx - Informationen
- 2xx - Erfolgreiche Operation
  - z.B. 200 OK
- 3xx - Umleitung
  - z.B. 301 Moved Permanently
- 4xx - Client-Fehler
  - z.B. 401 Unauthorized
  - z.B. 403 Forbidden
  - z.B. 404 Not Found
- 5xx - Server-Fehler
  - z.B. 500 Internal Server Error
  - z.B. 503 Service Unavailable

GET und POST Requests

- **GET Request**
  - Input-Daten in URL, d.h. Teil des GET Headers
  - z.B. `GET /index.html?page=120 HTTP/1.1`
  - Bookmarks möglich

- **POST Request**
  - Daten sind Teil des POST Bodys
  - z.B. oft bei Formularen genutzt
  - Browser fragt bei nochmaligem Senden nach

GET und POST Requests - Security

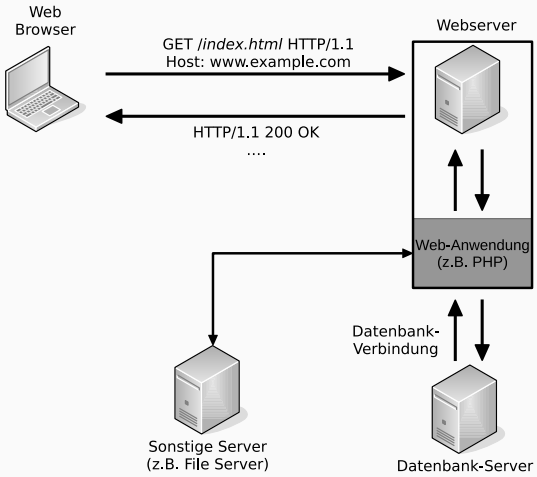
- **Manipulation leicht möglich**
  - GET Daten Manipulation über URL
  - POST Daten Manipulation mit Web-Debugging Tools, z.B. direkt in Firefox oder mit Burp

- **Senden beliebiger Eingabe-Daten möglich**
  - Muss nicht von Web-Applikation angeboten werden
  - Überprüfung auf Server-Seite unbedingt nötig

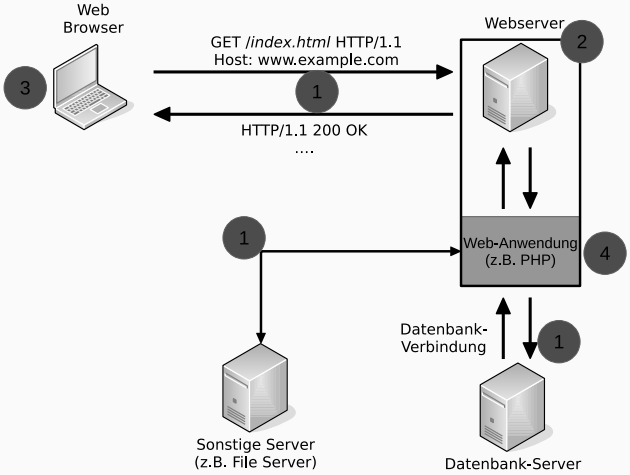
Cookies

- Ablage von beliebigen Werten auf Client-Seite
  - z.B. Inhalt des Warenkorbs
  - z.B. Authentifizierungsparameter
- Kann auf Client-Seite manipuliert werden
- Flags möglich
  - `Secure` - Cookie wird nur über HTTPS übertragen
  - `HttpOnly` - Cookie auf Client-Seite nicht zugreifbar
  - und mehr ...

Typische Web-Anwendungen



Typische Web-Anwendungen - Angriffsoberflächen



1) Kommunikation und Zugriffsschutz
2) Webserver-Software
3) Client-Sicherheit
4) Web-Anwendung

Absicherung der Kommunikation und Zugriffsschutz

Viele Web-Anwendungen vertrauen auf HTTPS.

Welcher Schutz wird dadurch erreicht?

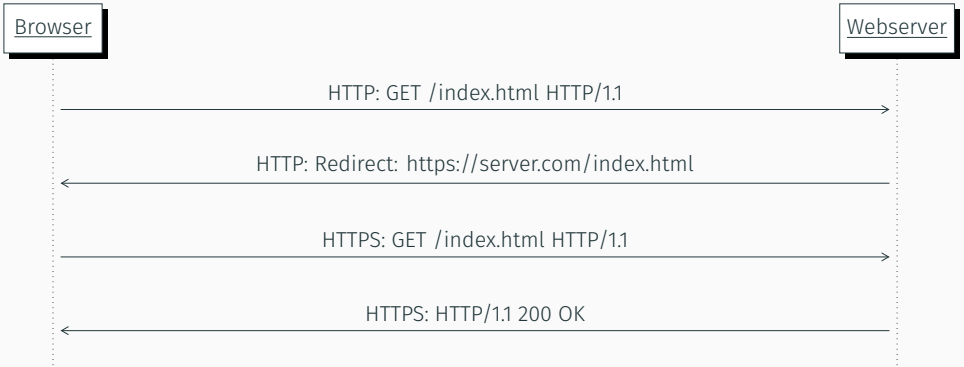
Wichtige TLS-Erweiterungen für HTTPS

- *Server Name Indication (SNI)*
  - Häufige Konfiguration: Ein Web-Server kann mehrere Web-Seiten beinhalten
  - Problem: Zertifikat für welche Web-Seite bei TLS-Verbindung mitschicken?
  - Lösung: SNI-Extension erlaubt Angabe von *server_name* in *Client Hello*

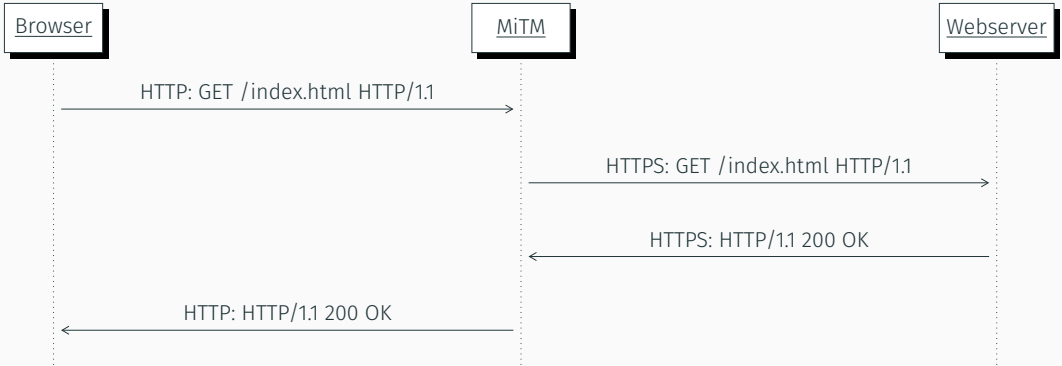
- Downgrade-Verhinderung der TLS Version: *Signaling Cipher Suite Value (SCSV)*
  - Problem:
    - Webserver unterstützen viele alte TLS-Versionen für alte Clients
    - Handshake nicht abgesichert. MiTM kann unterstützte Versionen reduzieren
  - Lösung:
    - Client erklärt höchste unterstützten Version in *TLS_FALLBACK_SCSV*-Wert
    - Server bricht Verbindung ab, wenn er höhere Version unterstützt *inappropriate_fallback*

SSL-Stripping

- Viele Webserver bieten HTTP (port 80) und HTTPS (port 443) an



SSL-Stripping: Durchführung des Angriffs



Maßnahmen gegen SSL-Stripping

- Durch den Nutzer: Bei der URL immer *https://* voranstellen
  - Schützt bei manuell eingegebenen URLs. Schützt nicht bei Verlinkungen
- Durch den Browser: Automatisches auswählen von HTTPS für bekannte Seiten
  - Benötigt Regeln welche Seiten HTTPs unterstützen (z.B. im Browser)
  - HTTPS Everywhere: Browser-Plugin der EFF mit Regelsätzen
- Durch Web-Seite: Mitliefern von *Host Strict Transport Security* (HSTS)
  - Seite setzt beim ersten Kontakt HTTP-Header:
    `Strict-Transport-Security: max-age="31536000"`
  - Browser wird für Seite ab diesem Zeitpunkt immer HTTPS verwenden
  - Gültig ab: Erster Besuch der Seite oder bei *HSTS preloading* in Browsern

Zugriffsschutz auf Sensitive Daten

- **Sensitive Daten dürfen nur nach erfolgreicher Authentifizierung ausgeliefert werden**
  - Daten auf den Seiten der Web-Anwendung
  - Zugriff auf Datenbanken und sonstige genutzte Dienste wie *File Server*
- **Schutz vor unerlaubtem Zugriff auf Datenbanken und sonstiger Dienste**
  - Netzwerksicherheit: Zugriff auf Dienste beschränken
  - Konfiguration der Web-Anwendung enthält Zugangsdaten für andere Dienste
- **Zugriffsschutz der Webseite selbst**
  - Web-Server: *Basic und Digest Authentication*, *TLS Client Authentication*
  - Web-Anwendung: Login-Formulare und Session-Management

Authentifizierung durch Web-Server

- **Basic Authentication**
  - Server fordert Authentifizierung für bestimmten Bereich an
  - Nutzername und Passwort werden Base64 kodiert
  - Kein Schutz von Vertraulichkeit und Integrität

- **Digest Access Authentication**
  - Server fordert Authentifizierung an und sendet Nonce
  - Client generiert Hash von Nutzer, Passwort, Nonce, HTTP Method, URI
  - Hashwert wird an Server geschickt und dort überprüft
  - Passwort wird somit nicht im Klartext übertragen

.htaccess

- Beispiel für .htaccess für HS Augsburg:

```
AuthBasicProvider ldap
AuthType Basic
AuthName "Ihren HS-Account"
require ldap-attribute employeeType=Angestellte
require ldap-attribute employeeType=Professoren
require ldap-attribute employeeType=Studenten
AuthLDAPURL ldaps://ldap1.hs-augsburg.de/ou=People,dc=fh-augsburg,dc=de
```

- Beispiel für htaccess für Ihre Eigene Homepage bei HS Augsburg
  - Link: https://www.hs-augsburg.de/Binaries/Binary25048/Beschreibung-eigeneHomepage.pdf

Authentifizierung durch Client-Zertifikate

- Gängige Webserver unterstützen Prüfung von TLS Client-Zertifikaten
  - Konfiguration eigener *Root Stores* möglich
  - Einschränkung durch Prüfung von Zertifikats-Feldern (z.B. *C*, *O*, *OU*, *CN*, ...)
- Beispiel in Apache2:

```
SSLVerifyClient      require
SSLCACertificateFile "conf/ssl.crt/ca.crt"

<Directory "/nur-fuer-studierende-professoren">
  SSLVerifyClient      require
  SSLOptions           +FakeBasicAuth
  SSLRequireSSL
  SSLRequire       %{SSL_CLIENT_S_DN_O}  eq "Hochschule Augsburg" \
                and %{SSL_CLIENT_S_DN_OU} in {"Professoren", "Studenten"}
</Directory>
```

Authentifizierung durch Web-Anwendung

- **Authentifizierung erfolgt durch programmierte Logik in der Anwendung**
  - Webserver “kennt” Login nicht sondern ist nicht eingebunden

- **Notwendige Voraussetzung: Session Management**
  - Web-Anwendung muss *Session-Cookie* mit *zufälligem* Wert generieren
  - Browser schickt *Session-Cookie* mit jeder Anfrage mit
  - Web-Anwendung muss Authentifizierungs-Status für *Session Cookie* speichern

- **Login erfolgt durch Übermittlung von Login-Daten in eigenem Formular**
  - Erfolgt meistens über *POST* Requests
  - Prüfung der Login-Daten durch Anwendung $\rightarrow$ Datenbank mit Login-Informationen notwendig

HTTP GET Requests können sehr einfach manipuliert werden während POST Requests nicht von der Client-Seite aus manipuliert werden können. Richtig oder falsch?

A) Richtig

B) Falsch

Bei der Basic Authentication über HTTPS werden Benutzername und Passwort im Klartext übertragen und sind somit für Man-in-the-Middle Angriffe anfällig. Richtig oder falsch?

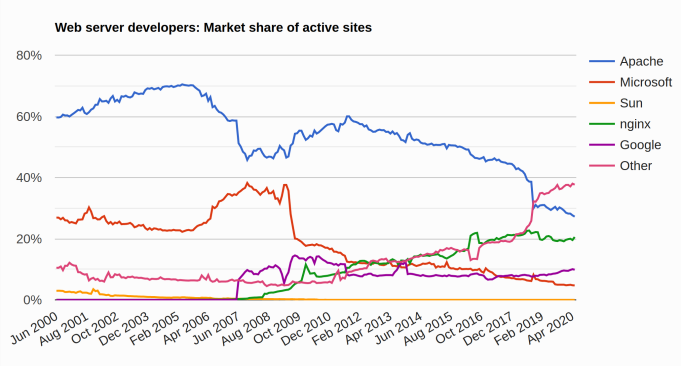
A) Richtig

B) Falsch

Sicherheit von Web-Servern

Web-Server im Internet

- **Web-Server nehmen HTTP-Anfragen entgegen und versenden HTTP-Antworten**
  - Ausführung von Scripten: externe Prozesse oder als eingebettete Module
  - Teilweise Frameworks zum Bau von Web-Anwendungen (z.B. Apache Struts)



Netcraft Studie: Web-Server bei aktiven Seiten im Internet. Stand Mai 2020 (Quelle)

Schwachstellen in Web-Server Software

- **Web-Server Software muss komplexe Protokolle verarbeiten**
  - Früher: Häufig Schwachstellen bei Kern-Funktionalität von Web-Servern
  - Heute: Tendenz zu kritischen Schwachstellen in komplexen Web-Frameworks

- **Einsatz von Web-Servern in typischen Internet-Szenarien**
  - Bewährte Server-Software verfügbar für gängige Betriebssysteme
  - Software-Updates durch Administratoren bei Schwachstellen sehr wichtig!

- **Einsatz von Web-Servern in eingebetteten Systemen**
  - Häufig Einsatz von unbekannten oder selbst entwickelten Web-Servern
  - Updates nur durch Firmware-Entwickler möglich → Hersteller in der Pflicht

Scannen von Web-Servern nach Schwachstellen

- **Erkennung von Fehler aufgrund von Schwachstellen und Fehlkonfigurationen**
  - Scanner sind Tools mit Modulen zur Erkennung häufiger Probleme
  - Beispiele: Fehlkonfiguration von TLS, alte Versionen mit Schwachstellen, ...

- **Beispiele für Scanner**
  - Nessus: https://de.tenable.com/products/nessus
  - OpenVAS https://www.openvas.org/index-de.html
  - Nikto: https://cirt.net/Nikto2
  - wpscan: https://wpscan.org/

Client Web-Security

Was stört Sie beim Besuch von Webseiten?

Was macht Ihnen dabei Sorgen?

Tracking / Einbindung unerwünschter Inhalte

- Problem
  - Verfolgung von Nutzern über Website-Grenzen hinweg
  - Unterstützt durch Cookies, Third-Party Scripts, Canvas Fingerprinting, etc.

- Auswirkung
  - Individuelle Verhaltensweisen werden beobachtet
  - Privatsphäre wird dadurch beeinträchtigt

- Mögliche Lösungen
  - Löschen aller Cookies nach Beenden des Browsers
  - Darstellung und Einschränkung von Third-Party Inhalten, z.B. mit Browser Erweiterungen *uBlock Origin* und *uMatrix*

Welches Problem könnten Ad-Blocker mit sich bringen?

Authentizität einer Website

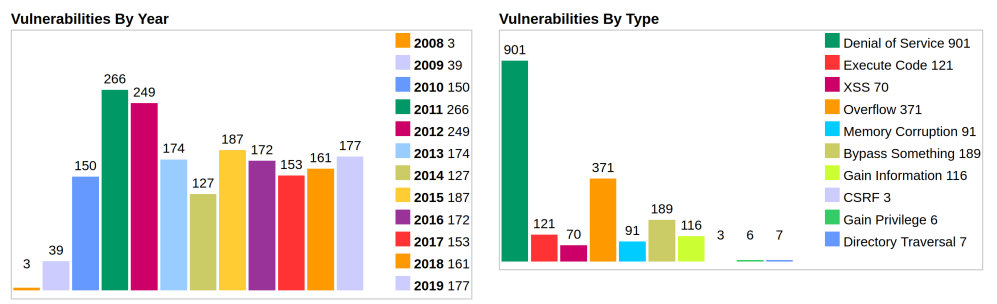
- **Problem**
  - Original Seite vs. Fake Seite (z.B. Phishing)
  - Optisch immer schwerer zu erkennen
- **Auswirkung**
  - Preisgabe von Daten an ”falschen” Empfänger
  - Ausführung von möglicherweise nicht vertrauenswürdigem Code
- **Mögliche Lösungen**
  - Auf korrekte Schreibweise der URL achten
  - Auf Symbole (z.B. grünes Schloss) achten und Warnungen ernst nehmen

Abhören bzw. Manipulation der Kommunikation

- **Problem**
  - Daten-Austausch mit Website kann von Netzknoten eingesehen werden
  - Auch eine Manipulation der Daten ist denkbar
- **Auswirkung**
  - Identitätsdiebstahl, Überwachung, und vieles mehr ...
- **Mögliche Lösung**
  - Auf `https://` in URL achten
  - Browser Erweiterung *HTTPS Everywhere*

- **Browser müssen sehr komplexe Webseiten verarbeiten und darstellen**
  - HTML5 bietet Beschreibungssprache mit sehr vielen Funktionalitäten
  - Browser müssen mit teilweise fehlerhaften Dokumenten umgehen
  - Angreifer können durch JavaScript eigenen Code im Browser ausführen

  → Hohes Potential für ausnutzbare Schwachstellen



Anzahl Schwachstellen in Google Chrome, Stand Juni 2020 (Quelle)

Sehr wichtig:
Regelmäßige Browser Updates!

Gibt es noch Fragen?