esse

Einführung in Security – VO 12: Web Application Security

Florian Fankhauser, Christian Schanes, Rafael Vrecar

2024W

ESSE (Establishing Security) – IT Security Research Team
Research Group for Industrial Software (INSO)
https://establishing-security.at/

Agenda

- Einführung in Web Application Security
  - Typische Komponenten einer modernen Web-Applikation
  - Arten von Web-Servern
  - Hyper Text Transfer Protocol (HTTP)
    - Aufbau
    - Session Management
    - Authentifizierung
  - Beispiele für Sicherheitslücken
- Ausblick: Browser Security
- Literatur, Zusammenfassung

Einführung in Web Application Security

- Vor vielen Jahren waren Websites noch einfach – nahezu keine Interaktion, keine Formulare, statische Seiten
- Heute: Steigende Anzahl von neu entwickelten, komplexen Web-Applikationen, Single Page Websites, viel Interaktion
- Mailclients (z.B. Roundcube, Horde), Studierendenverwaltungssysteme (z.B. TISS), Cloud-basierte Speicherdienste (z.B. Dropbox, SkyDrive), Social Media...
- Großer Funktionsumfang/Hohe Komplexität bei Web-Applikationen, z.B. durch JavaScript, HTML5
- Web-Technologie nicht nur im Web – auch z.B. bei Apps, auf SIP-Telefonen etc.
- Hohe Anzahl an unterschiedlichen, unkontrollierbaren Clients

Typische Komponenten einer modernen Web-Applikation

- Webserver
- Uniform Resource Locator (URL) inkl. Protokoll, evtl. Username/Passwort, Server-Adresse, Server-Port, Pfad, Query String, incl. Encoding
- Hyper Text Transfer Protocol (HTTP)
- Hyper Text Markup Language (HTML) in unterschiedlichen Versionen mit unterschiedlichem Funktionsumfang
- Cascading Style Sheets (CSS)
- Client-Side-Scripting (z.B. JavaScript)

(Vergleiche siehe auch Zalewski, The Tangled Web)

Weitere Komponenten einer modernen Web-Applikation

- Nicht-HTML-Daten, z.B. Bilder, Videos, Extensible Markup Language (XML)
- Unterschiedliche Browser (z.B. Seamonkey, Firefox, Chromium, Safari, Edge) auf unterschiedlichen Plattformen (PC, Tablet, Smartphone,...)
- Plugins/Addons

(Vergleiche Zalewski, The Tangled Web)

Arten von Web-Servern

- Statische Web-Server: Anzeige statischer Inhalte (ursprüngliche Idee des WWW)
- Dynamische Web-Server:
  - Statische und dynamische Inhalte
  - Oftmals Anbindung an möglicherweise mehrere Datenbanken oder File-Server
  - Verschiedene Web-Applikationen können auf einem Server in Betrieb sein
- Reverse-Proxy: serviciert oft mehrere Web-Server, befindet sich zwischen Client und Web-Server
- Beispiele für Web-Server: Apache HTTP Server, Apache Tomcat, Microsoft IIS, lighttpd, nginx

Beispiel eines HTTP-GET-Requests

```
GET / HTTP/1.1
Host: security.inso.tuwien.ac.at
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:28.0) Gecko\
    /20100101 SeaMonkey/2.25
Accept: text/html,application/xhtml+xml,application/xml;q\
    =0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.7,de;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
```

Beispiel einer HTTP-Response

```
HTTP/1.1 200 OK
Date: Thu, 03 Apr 2014 16:30:17 GMT
Server: Apache
Last-Modified: Thu, 03 Apr 2014 06:19:05 GMT
ETag: "f-13df-4f61d60c74840"
Accept-Ranges: bytes
Content-Encoding: gzip
Content-Length: 2033
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

HTTP-Request-Methoden GET, POST, PUT, DELETE

- GET: Parameter werden in der URL mitgegeben
- POST: Parameterübergabe mittels HTTP-Payload
- PUT: ermöglicht Datenupload am Server
- DELETE: Löschen von Server-Ressourcen

(Vergleiche `https://tools.ietf.org/html/rfc2616`)

HTTP-Request-Methoden OPTIONS, HEAD, TRACE

- OPTIONS: Abfragen der HTTP-Methoden, die der Server unterstützt
- HEAD: Retourniert nur HTTP-Header
- TRACE: Alle Änderungen (z.B. durch Reverse-Proxy) am HTTP-Request nachvollziehbar

- Best Practise: Sämtliche nicht verwendete HTTP-Methoden abschalten, vor allem OPTIONS, TRACE, PUT, DELETE

(Vergleiche `https://tools.ietf.org/html/rfc2616`)

HTTP-Session-Management

HTTP ist stateless → Session-Management erfolgt z.B. über Cookies, Hidden-Form-Fields, URL-Parameter, HTTP Header Tokens

- Client-seitige Sessions vs. Server-seitige Sessions
- HTTP-Header: Cookie oder Authorization-Header (z.B. non-persistent Cookies, JSON Web Tokens)
- URL-Parameter: in der URL angegeben mit `"parameter=wert"`, mehrere Parameter mit `&` separiert
- Hidden-Form-Fields: nicht sichtbar im HTML-Code, durch Source-Code-Analyse auffindbar (`<input type="hidden" name="Name" value="Wert">`)
- Angreifer:innen können Session-IDs möglicherweise ausspionieren und manipulieren (z.B. Session-Analyse)

HTTP-Authentifizierung – Basic

- Authentifizierung erfolgt über Benutzer:innenname und Passwort, separiert mittels ":"
- Verwendung von Base64-Encoding
- Beispiel:
  - Server: `WWW-Authenticate: Basic realm="ESSE"`
  - Client: `Authorization: Basic c2VjdXJlLXVzZXI6ZXNzZQ==`
  - `$ echo "c2VjdXJlLXVzZXI6ZXNzZQ=="|base64 -d`
    `secure-user:esse`
- Credentials werden (so gut wie) im Klartext übertragen!

(Vergleiche `https://tools.ietf.org/html/rfc2617`)

HTTP-Authentifizierung – Digest: Beispiel

- Authentifizierung erfolgt über Digesting der Credentials mittels MD5
- *Server*

```
WWW-Authenticate: Digest realm="ESSE",
nonce="tNO/HCj2BAA=8fd115d3a77df5be9b6008c1cbc1a9f22855559b",
algorithm=MD5, domain="/rest/", qop="auth"
```

- *Client*

```
Authorization: Digest username="secure-user", realm="ESSE",
nonce="tNO/HCj2BAA=8fd115d3a77df5be9b6008c1cbc1a9f22855559b",
uri="/rest/", algorithm=MD5,
response="eb88d1862f75c82ebe60304d5a3780c5", qop=auth,
nc=00000001,cnonce="0dbe5538b34a56b1"
```

(Vergleiche `https://tools.ietf.org/html/rfc2617`)

HTTP-Authentifizierung – Digest: Berechnung

- HA1 = echo -n "secure-user:ESSE:esse"|md5sum
  → b1e68fa8b6a0ab3d4833ab0a3c54e02f
- HA2 = echo -n "GET:/rest/"|md5sum
  → f7c603b6955bd38ef4a3283e89c57f7d
- Response = echo -n
  "HA1:nonce:nonceCount:clientNonce:qop:HA2"|md5sum
- Response = echo -n
  "b1e68fa8b6a0ab3d4833ab0a3c54e02f:tNO/HCj2BAA=8fd115d3a77
  df5be9b6008c1cbc1a9f22855559b:00000001:0dbe5538b34a56b1:auth:
  f7c603b6955bd38ef4a3283e89c57f7d"|md5sum
  → eb88d1862f75c82ebe60304d5a3780c5

(Vergleiche https://tools.ietf.org/html/rfc2617)

Beispiele für Cookies

```
domain / readable by all machines in a given domain / path / secure /
     expiration / name / value
www.tuwien.ac.at    FALSE / FALSE 0
    fe_typo_user  d23c126a081555a51fc9a13e076b93fc
iu.zid.tuwien.ac.at FALSE / TRUE  0
    apps          -76-
iu.zid.tuwien.ac.at FALSE / TRUE  1428101315
    zidToken      af5e513d98e5fe3b54393d72bbfafe58ad5a2672.394c305.3366761
tiss.tuwien.ac.at   FALSE / FALSE 0
    TISS_AUTH     e8c9c83ecbad4238b3a4cff00a8e3dbd4dbeef3c9510951b390b62ea4bbf368b
tiss.tuwien.ac.at   FALSE / FALSE 1427406716
    _tiss_session 51c639f6685f053f78df77b9f4e14c40
$ curl -s -I 'https://tiss.tuwien.ac.at/' | grep -i cook
Set-Cookie: TISS_LANG=de; path=/; expires=Sat, 26-Mar-2016 20:58:33 GMT
Set-Cookie: _tiss_session=f63a533dc1db1e1c178156d779e58346; path=/;
     expires=Thu, 26-Mar-2015 22:28:33 GMT; HttpOnly
```

JWT: JSON Web Token

- Spezifiziert in RFCs
  - RFC 7519: JSON Web Token (JWT)
  - RFC 7515: JSON Web Signature (JWS)
- Aufbau
  - Header
  - Payload
  - Signatur
- Vordefinierte und eigene Claim Namen bei Payload
- Verwendung in Cookies oder als Authorization-Header bei HTTP mit eigenen Vor- und Nachteilen

Beispiel für ein JWT: JSON Web Token

- Header: `{"typ":"JWT", "alg":"HS256"}` → JSON Web Token, Hash-based Message Authentication Code (HMAC) SHA-256
- Payload/JWT Claims: `{"iss":"joe", "exp":1551984189, "http://example.invalid/is_root":true}`
- Signatur: `HMACSHA256(BASE64URL(UTF8(Header)) + '.' + BASE64URL(UTF8(Payload)), secret-key)`
- `eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9.eyJpc3Mi OiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6L y9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ.dBjftJeZ4CV P-mB92K27uhbUJU1p1r_wW1gFWFOEjXk`

(Vergleiche RFC 7519, RFC 7515)

OWASP Top 10 2013

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function Level Access Control
- A8: Cross Site Request Forgery (CSRF)
- A9: Using Components With Known Vulnerabilities
- A10: Unvalidated Redirects and Forwards

(Vergleiche `https://owasp.org/www-pdf-archive/OWASP_Top_10_-_2013.pdf`)

OWASP Top 10 2017

- **A1: Injection**
- **A2: Broken Authentication**
- **A3: Sensitive Data Exposure**
- A4: XML External Entities
- **A5: Broken Access Control**
- **A6: Security Misconfiguration**
- A7: Cross Site Scripting (XSS)
- A8: Insecure Deserialization
- **A9: Using Components With Known Vulnerabilities**
- A10: Insufficient Logging & Monitoring

(Vergleiche `https://owasp.org/www-project-top-ten/2017/`)

OWASP Top 10 2021

- **A01 Broken Access Control**
- **A02 Cryptographic Failures**
- **A03 Injection**
- A04 Insecure Design
- **A05 Security Misconfiguration**
- **A06 Vulnerable and Outdated Components**
- **A07 Identification and Authentication Failures**
- A08 Software and Data Integrity Failures
- **A09 Security Logging and Monitoring Failures**
- A10 Server Side Request Forgery (SSRF)

(Vergleiche `https://owasp.org/Top10/`)

Beispiele für Sicherheitslücken

- Aus den OWASP Top 10-Sicherheitslücken werden nun ausgewählte Sicherheitslücken vorgestellt
- Auch „alte“ Sicherheitslücken müssen nicht unbedingt geschlossen sein nach Jahren (z.B. CVE-2011-2461)
- Meistens gibt es weitere Gefahren/Maßnahmen als die, die heute vorgestellt werden
- Weitere Sicherheitslücken und auch fortgeschrittenere Formen für vorgestellte Sicherheitslücken vorhanden
- $\rightarrow$ weitere LVAs der ESSE besuchen :)

Injection-Angriffe – Beschreibung und Gefahren

- **Vorgehen:** Kommandos werden an einen Interpreter geschickt und in einer Art und Weise ausgeführt wie es nicht vorgesehen ist
- SQL-Injection, Command-Injection, Template-Injection, XML-Injection, XPath-Injection etc.

- **Gefahren**, z.B.:
  - Datenbankeinträge können manipuliert, ausgelesen oder gelöscht werden
  - Unautorisierte Zugriffe auf das Betriebsystem sind möglich
  - Verursachung von Software-Abstürzen, DOS
  - Port-Scanning

Injection-Angriffe – Beispiele für Maßnahmen

- Alle Eingaben validieren!
  - Denylists (Blacklists)
  - (besser) Allowlists (Whitelists)
- Verwendung von Prepared-Statements
- Vermeidung von Benutzer:inneneingaben in Templates
- Minimale Zugriffsrechte pro Benutzer:in

XML-Injection

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE accounts SYSTEM "accounts.dtd">
<accounts>
    <account>
        <username>e123456</username>
        <pwd>hash(deem*ohG1oor)</pwd>
        <perms>r</perms>
    </account>
</accounts>
```

Durchgeführte XML-Injection

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE accounts SYSTEM "accounts.dtd">
<accounts>
    <account>
        <username>e123456</username><pwd>hash(deem*ohG1oor)</pwd>
        <perms>rw</perms></account><account><username>e654321
        </username>
        <pwd>deem*ohG1oor</pwd>
        <perms>r</perms>
    </account>
</accounts>
```

XPath-Injection

- XPath dient zum Zugriff auf ein XML-Dokument
- XPath-Injection ist ähnlich der SQL-Injection

```
authenticated = "//account[username/text()='" +
    $username +
    "' AND pwd/text()='" + hash($pwd) + "']"
```

XPath-Injection

- XPath dient zum Zugriff auf ein XML-Dokument
- XPath-Injection ist ähnlich der SQL-Injection

```
authenticated = "//account[username/text()='" +
    $username +
    "' AND pwd/text()='" + hash($pwd) + "']"
```

```
Username: attack' OR 1=1 OR '1'='1
Passwort: successful
```

XPath-Injection

- XPath dient zum Zugriff auf ein XML-Dokument
- XPath-Injection ist ähnlich der SQL-Injection

```
authenticated = "//account[username/text()='" +
    $username +
    "' AND pwd/text()='" + hash($pwd) + "']"
```

```
Username: attack' OR 1=1 OR '1'='1
Passwort: successful
```

```
authenticated = "//account[username/text()='attack' OR 1=1
    OR
    '1'='1' AND pwd/text()='hash(successful)']"
```

Cryptographic Failures

- Durch Fehler im Bereich von Kryptographie oder unzureichende Kryptographie führen oft zur Preisgabe von Informationen
- **Gefahr:** Angreifer:in hat Zugriff auf sensible Daten
- **Maßnahmen:**
  - Bewerten von Informationsobjekten (Risiko-/Bedrohungsanalyse)
  - Verhinderung der Speicherung nicht erforderlicher Daten
  - Verschlüsselung mit sicheren Algorithmen
  - Verwendung komplexer Schlüssel (z.B. 2048 Bit) bzw. Salt
  - Verschlüsselte Daten und Schlüssel separat speichern
  - Schlüssel schützen

Insufficient Transport Layer Protection

- *Siehe auch Vorlesungseinheit Netzwerksicherheit*
- Unverschlüsselte Übertragung von (sensiblen) Inhalten
- Moderne Browser zeigen unverschlüsselte Websites als unsicher an
- **Gefahr:** Angreifer:in kann vertrauliche Informationen mitlesen oder verändern (z.B. Kreditkartennummern, Passwörter)
- **Maßnahmen:**
  - Verwendung von Transport Layer Security (TLS)
  - Verschlüsseln und Signieren der Daten vor der Übertragung
  - Setzen des Secure-Flags bei Cookies
  - HTTP Strict Transport Security (HSTS)

Überlegen Sie, warum

abgelaufene Zertifikate

ein Sicherheitsproblem darstellen (können)?

Identification and Authentication Failures/Broken Authentication and Session Management

- Zur Erinnerung: HTTP ist ein stateless Protokoll, das Session-Informationen verwendet, um Statusinformationen zu speichern

- **Gefahr:**
  - Session-Hijacking (z.B. Firesheep)
  - Benutzer:innen-Accounts übernehmen inkl. Funktionen wie z.B. Passwort ändern, Sicherheitsfrage, Passwort vergessen, etc.
  - Unsichere Session-Verwaltung (z.B. unzureichend zufällige Session IDs)

Identification and Authentication Failures – Maßnahmen

- Verwendung von TLS und Sicherstellung, dass die Session-ID durch TLS geschützt ist
- Gute, zufällige Session-IDs
- Sicherstellen, dass ein Logout wirklich die Session terminiert
- Logout prinzipiell mittels Logout-Button durchführen, nicht nur Browser schließen
- Logout nach passender, definierter inaktiver Zeit von Benutzer:innen

Wo sind Ihnen schon einmal

Identification and Authentication Failures *begegnet?*

Failure to Restrict URL Access (Broken Access Control)

- **Typische Fehler:**
  - Es werden nur bestimmte Links und Menü-Punkte angezeigt, die anderen werden versteckt (Security by Obscurity)
  - Presentation Layer Access Control: Keine Prüfung am Server
- **Gefahr:**
  - Angreifer:in ruft Funktionen auf, für die nicht berechtigt
  - `www.insecure-site.invalid/[user|admin]/getAccntData`
- **Maßnahmen:**
  - Für jede URL sicherstellen, dass Benutzer:innenzugriff erlaubt
  - Auf bestimmten Seiten Zugriff verbieten (z.B. Config-Dir)
  - Rollenbasiertes Zugriffsmodell
- $\rightarrow$ Missing Function Level Access Control

Insecure Direct Object References (Broken Access Control)

- Ähnlich zu *Failure to Restrict URL Access*
- **Typische Fehler:**
  - Es wird direkt über eine vorhersehbare ID auf Objekte zugegriffen
  - Es werden nur bestimmte Objekte angezeigt, die anderen werden versteckt
  - Kontrolle nur am Presentation-Layer: Keine Überprüfung auf Server-Seite
- **Gefahr:**
  - Angreifer:in wird ähnliche Nummern ausprobieren:
  - www.insecure-server.invalid?account=100
  - www.insecure-server.invalid?account=102

Insecure Direct Object References: Maßnahmen

- Keine direkte Referenz auf Objekte
- Verwendung von zufälligen, temporären Mapping-Values, z.B.
  - Statt ?account=100 → ?account=8w9e9fgi
  - Statt ?file=accounting.xls → ?file=119347
- Überprüfung der Objekt-Referenzen auf: Formatierung, ob Benutzer:in die Berechtigung hat zuzugreifen, etc.

Unvalidated Redirects and Forwards

- Häufig erfolgen Weiterleitungen von Wep-Applikationen zu anderen Webseiten, Zieladresse wird oft unter Verwendung von Benutzer:inneneingaben bestimmt
- Gefahr:
  - Angreifer:in kann dadurch das Opfer auf Malware- oder Phishing-Seiten weiterleiten
  - http://example.invalid/redirect.php?url=http://evil.invalid
  - Umgehung von Autorisierungsmechanismen
- Maßnahmen:
  - Möglichst keine URLs bestehend aus Benutzer:innen-Parametern für Redirects verwenden
  - Eingabevalidierung

Wo sind Ihnen schon einmal

Unvalidated Redirects and Forwards

begegnet?

Security Misconfiguration

- Sicherheit von Web-Applikationen hängt auch von darunterliegenden Software-Schichten ab, z.B. aktuelle Patches, Services, Ports, etc.
- **Gefahr:**
  - Ausnützen von Sicherheitslücken, da keine Patches installiert
  - Default-Accounts aktiv, Standard-Passwörter
  - Auslesen und Verändern von Daten, DoS
  - Fehlende Härtungen
  - Kein Senden von Security-Headern
- **Maßnahmen:**
  - Hardening-Prozess definieren
  - Regelmäßiges Scanning und Auditing der Konfiguration(en)
  - Sicheres Softwaredesign

Wo sind Ihnen schon einmal

Security Misconfigurations

begegnet?

Vulnerable and Outdated Components

- Immer wieder Sicherheitslücken in unterschiedlichster Software – offensichtlich auch Software-Komponenten, die im Web verwendet werden
- Gefahr:
  - Bekannte Sicherheitslücken können auf Grund oft freier Verfügbarkeit (unbemerkt) ausgenutzt werden – für unterschiedlichste Angriffe
- Maßnahmen:
  - Security im gesamten Lebenszyklus berücksichtigen – Analyse, Design, Implementierung, Test, Betrieb $\rightarrow$ zeitnahe Installation von Patches

Local File Inclusion (LFI)

- File Inclusion (FI) ermöglicht es Angreifer:innen, Datei einzubinden
- nutzen idR. in Zielanwendung implementierten Mechanismus zur „dynamischen Dateieinbindung“
- Local FI (LFI): Einbindung von bereits auf Server vorhandener Datei
- einmal mehr: mangelhafte Input-Validierung!
- **Gefahr:**
  - Inhalt der Datei anzeigen, aber auch
  - Codeausführung auf Webserver oder auch Client-Seite (z.B. JS)
  - Denial of Service (DoS)
  - Offenlegung sensibler Informationen

(Vergleiche `https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_Local_File_Inclusion`)

Google Hacking

- Über Google können Webseiten mit Schwachstellen ermittelt werden
- Websites mit potenziellen MySQL SQL Injection

```
inurl:"php?id" "You have an error in your SQL syntax"
```

- Oracle-Applikationen ermitteln

```
intitle:iSQL intitle:Release inurl:isqlplus
```

- Fehlerhafte Konfigurationen

```
filetype:inc intext:mysql_connect
```

- Web Cams

```
intitle:axis intitle:"video server"
```

- Jenkins mit offener Admin-Schnittstelle

```
intitle:"Dashboard [Jenkins]" intext:"Manage Jenkins"
```

- Weitere Suchbegriffe siehe auch

```
https://www.exploit-db.com/google-hacking-database/
```

Ausblick: Browser Security

- Immer mehr Funktionalität, z.B. VoIP, Video,...
- Sicherheitslücken durch Software (Patches, Plugins, ...)
- Unterschiedliche Browser reagieren unterschiedlich auf gleiche Anfragen
- Beispiele für Sicherheitsmechanismen von Browsern
  - Same-Origin
  - Security Policies für Cookies
  - Blocken von Ports (z.B. Port 110)
  - XSS-Schutz
- Sandboxen
- Zugriff auf lokale Files

Tools & Programmiersprachen für Sicherheitstests (Bsp.)

- Python
- Metasploit
- curl
- Web Developer
- Tamper Data
- Burp
- sqlmap
- Nikto
- DirBuster

Diese Tools können Ihnen auch im Rahmen von *Lab1* von Nutzen sein.

Tipp: Ein paar weitere wichtige Schlüsselwörter, die Sie brauchen könnten: Insecure Direct Object Reference (IDOR), Remote Code Execution (RCE)

Überlegen Sie: Zu welchen heute vorgestellten/gelisteten Sicherheitslücken passen die erwähnten Schlüsselwörter?

Literaturhinweise/Links – 1/2

- https://www.owasp.org/
  - OWASP Top 10 Project
  - OWASP Testing Guide
  - OWASP Web Goat Project
- Martin Burkhart. Gut behütet: OWASP API Security Top 10, 2020.
  https://heise.de/-4660904
- Michal Zalewski. *The Tangled Web: A Guide to Securing Modern Web Applications*.
  No Starch Press, San Francisco, CA, USA, 1. Auflage, 2011.
  ISBN 1593273886, 9781593273880
- Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, und Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, 1999.
  https://www.ietf.org/rfc/rfc2616.txt

Literaturhinweise/Links – 2/2

- John Franks, Phillip M. Hallam-Baker, Jeffery Hostetler, Scott D. Lawrence, Paul J. Leach, Ari Luotonen, und Lawrence C. Stewart. HTTP Authentication: Basic and Digest Access Authentication, 1999.
  `https://www.ietf.org/rfc/rfc2617.txt`
- Michael B. Jones, John Bradley, und Nat Sakimura. JSON Web Token (JWT), 2015b.
  `https://tools.ietf.org/rfc/rfc7519.txt`
- Michael B. Jones, John Bradley, und Nat Sakimura. JSON Web Signature (JWS), 2015a.
  `https://tools.ietf.org/rfc/rfc7515.txt` Shodan Search Engine

Zusammenfassung

- Grundlagen von Web-Applikationen
- Hyper Text Transfer Protocol (HTTP)
  - Protokoll
  - Authentifizierung
  - Session Management
- Unterschiedliche Sicherheitslücken basierend auf OWASP Top 10, z.B.
  - Broken Authentication and Session Management
  - Insecure Direct Object References
  - Failure to Restrict URL Access
- Google Hacking, Browser Security
- IT-Sicherheit kann nicht einzeln betrachtet werden

Vielen Dank!

https://establishing-security.at/