TECHNISCHE UNIVERSITÄT DRESDEN

Professur Datenschutz und Datensicherheit

Department of Computer Science, Institute for Systems Architecture, Chair of Privacy and Data Security

Pentestlab — Schwachstellen von Web-Anwendungen

dud.inf.tu-dresden.de

Stefan Köpsell (stefan.koepsell@tu-dresden.de)

Literatur

Mark Dowd, John McDonald, Justin Schuh: „The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities“

Hanqing Wu, Liz Zhao: „Web Security“

Prakhar Prasad: „Mastering Modern Web Penetration Testing“

Marcus Pinto, Dafydd Stuttard: “The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, 2nd Edition”

OWASP
The Open Web Application Security Project

https://www.owasp.org/

- internationale Vereinigung / Community
  - Entwicklung sicherer Anwendungen

- OWASP Foundation
  - Non-profit Organisation
  - gegründet 1. Dezember 2001
  - formaler OWASP Rahmen

OWASP
The Open Web Application Security Project

OWASP Dresden Stammtisch

https://www.owasp.org/index.php/OWASP_German_Chapter_Stammtisch_Initiative/Dresden

- Treffen alle 1 bis 2 Monate
- Vorträge zu IT-Sicherheit / Entwicklung sicherer Anwendungen
- breites Themenspektrum
- kostenfrei…

Top 10 Web Security Risks 2010 / 2013

OWASP
The Open Web Application Security Project

OWASP Top 10 – 2010 (old)	OWASP Top 10 – 2013 (New)
2010-A1 – Injection	2013-A1 – Injection
2010-A2 – Cross Site Scripting (XSS)	2013-A2 – Broken Authentication and Session Management
2010-A3 – Broken Authentication and Session Management	2013-A3 – Cross Site Scripting (XSS)
2010-A4 – Insecure Direct Object References	2013-A4 – Insecure Direct Object References
2010-A5 – Cross Site Request Forgery (CSRF)	2013-A5 – Security Misconfiguration
2010-A6 – Security Misconfiguration	2013-A6 – Sensitive Data Exposure
2010-A7 – Insecure Cryptographic Storage	2013-A7 – Missing Function Level Access Control
2010-A8 – Failure to Restrict URL Access	2013-A8 – Cross-Site Request Forgery (CSRF)
2010-A9 – Insufficient Transport Layer Protection	2013-A9 – Using Known Vulnerable Components (NEW)
2010-A10 – Unvalidated Redirects and Forwards (NEW)	2013-A10 – Unvalidated Redirects and Forwards
3 Primary Changes:	▪ Merged: 2010-A7 and 2010-A9 -> 2013-A6
▪ Added New 2013-A9: Using Known Vulnerable Components	▪ 2010-A8 broadened to 2013-A7

Mapping from 2010 to 2013 Top 10

OWASP
The Open Web Application Security Project

OWASP Top 10 – 2010 (old)	OWASP Top 10 – 2013 (New)
2010-A1 – Injection	2013-A1 – Injection
2010-A2 – Cross Site Scripting (XSS)	2013-A2 – Broken Authentication and Session Management
2010-A3 – Broken Authentication and Session Management	2013-A3 – Cross Site Scripting (XSS)
2010-A4 – Insecure Direct Object References	2013-A4 – Insecure Direct Object References
2010-A5 – Cross Site Request Forgery (CSRF)	2013-A5 – Security Misconfiguration
2010-A6 – Security Misconfiguration	2013-A6 – Sensitive Data Exposure
2010-A7 – Insecure Cryptographic Storage	2013-A7 – Missing Function Level Access Control
2010-A8 – Failure to Restrict URL Access	2013-A8 – Cross-Site Request Forgery (CSRF)
2010-A9 – Insufficient Transport Layer Protection	2013-A9 – Using Known Vulnerable Components (NEW)
2010-A10 – Unvalidated Redirects and Forwards (NEW)	2013-A10 – Unvalidated Redirects and Forwards
3 Primary Changes:	▪ Merged: 2010-A7 and 2010-A9 -> 2013-A6
▪ Added New 2013-A9: Using Known Vulnerable Components	▪ 2010-A8 broadened to 2013-A7

2013-A2 – Broken Authentication and Session Management

OWASP
The Open Web Application Security Project

HTTP is a "stateless" protocol

- Means credentials have to go with every request
- Should use SSL for everything requiring authentication

Session management flaws

- SESSION ID used to track state since HTTP doesn't
  - and it is just as good as credentials to an attacker
- SESSION ID is typically exposed on the network, in browser, in logs, …

Beware the side-doors

- Change my password, remember my password, forgot my password, secret question, logout, email address, etc…

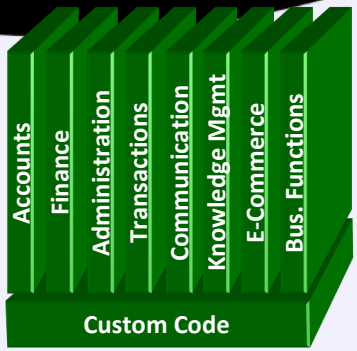
Typical Impact

- User accounts compromised or user sessions hijacked

Broken Authentication Illustrated

OWASP
The Open Web Application Security Project

1 User sends credentials

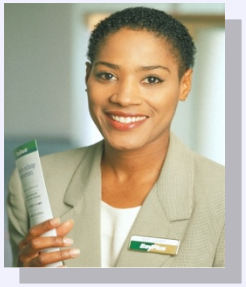


www.boi.com?JSESSIONID=9FA1DB9EA...

Site uses URL rewriting

(i.e., put session in URL)

2

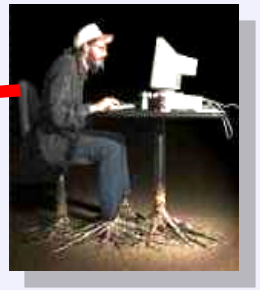


3 User clicks on a link to http://www.hacker.com in a forum

Hacker checks referer logs on www.hacker.com

and finds user's JSESSIONID

4



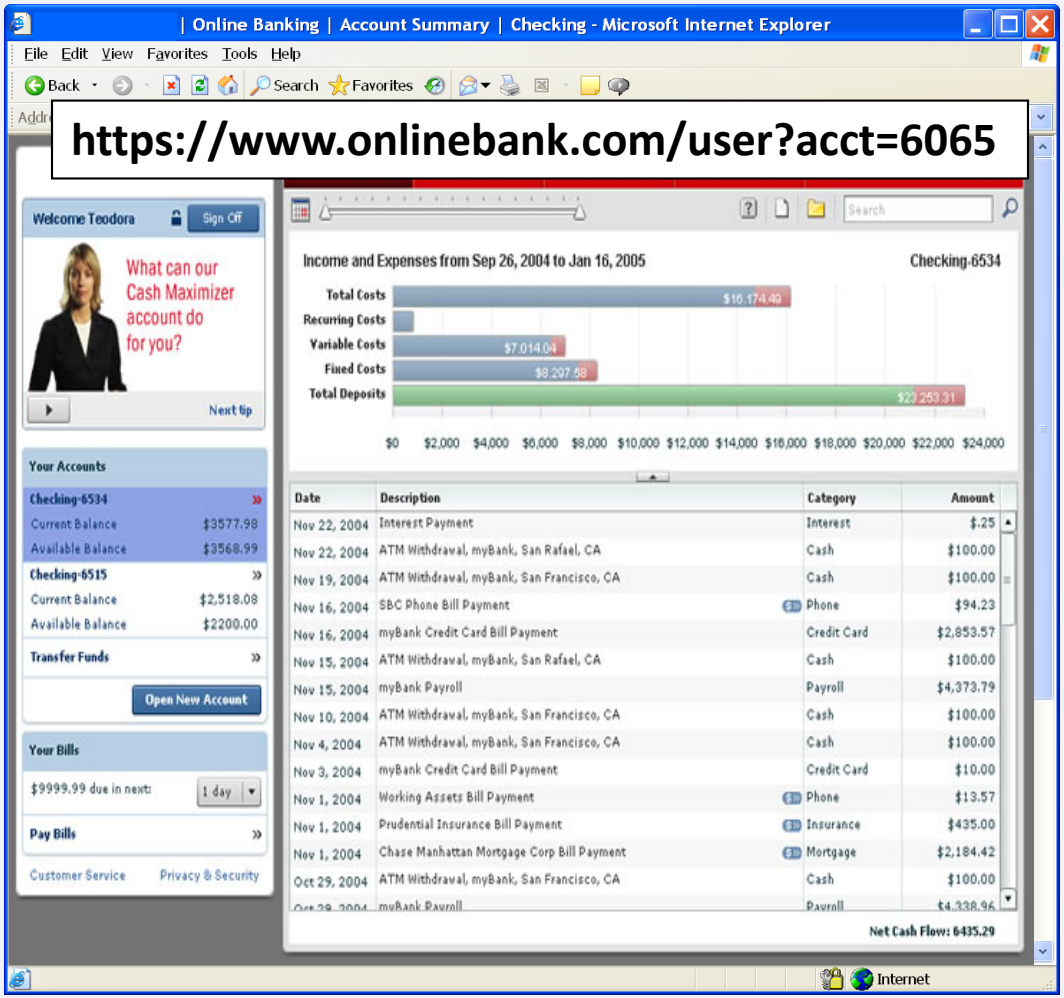
5 Hacker uses JSESSIONID and takes over victim's account

Mapping from 2010 to 2013 Top 10

OWASP Top 10 – 2010 (old)	OWASP Top 10 – 2013 (New)
2010-A1 – Injection	2013-A1 – Injection
2010-A2 – Cross Site Scripting (XSS)	2013-A2 – Broken Authentication and Session Management
2010-A3 – Broken Authentication and Session Management	2013-A3 – Cross Site Scripting (XSS)
2010-A4 – Insecure Direct Object References	2013-A4 – Insecure Direct Object References
2010-A5 – Cross Site Request Forgery (CSRF)	2013-A5 – Security Misconfiguration
2010-A6 – Security Misconfiguration	2013-A6 – Sensitive Data Exposure
2010-A7 – Insecure Cryptographic Storage	2013-A7 – Missing Function Level Access Control
2010-A8 – Failure to Restrict URL Access	2013-A8 – Cross-Site Request Forgery (CSRF)
2010-A9 – Insufficient Transport Layer Protection	2013-A9 – Using Known Vulnerable Components (NEW)
2010-A10 – Unvalidated Redirects and Forwards (NEW)	2013-A10 – Unvalidated Redirects and Forwards
3 Primary Changes:	▪ Merged: 2010-A7 and 2010-A9 -> 2013-A6
▪ Added New 2013-A9: Using Known Vulnerable Components	▪ 2010-A8 broadened to 2013-A7

Insecure Direct Object References Illustrated

OWASP
The Open Web Application Security Project



- **Attacker notices his acct parameter is 6065**

  ?acct=6065

- **He modifies it to a nearby number**

  ?acct=6066

- **Attacker views the victim's account information**

Indirekte Objekt-Referenzierung

Mapping zwischen externer und interner Referenz

HTTP-Anfrage

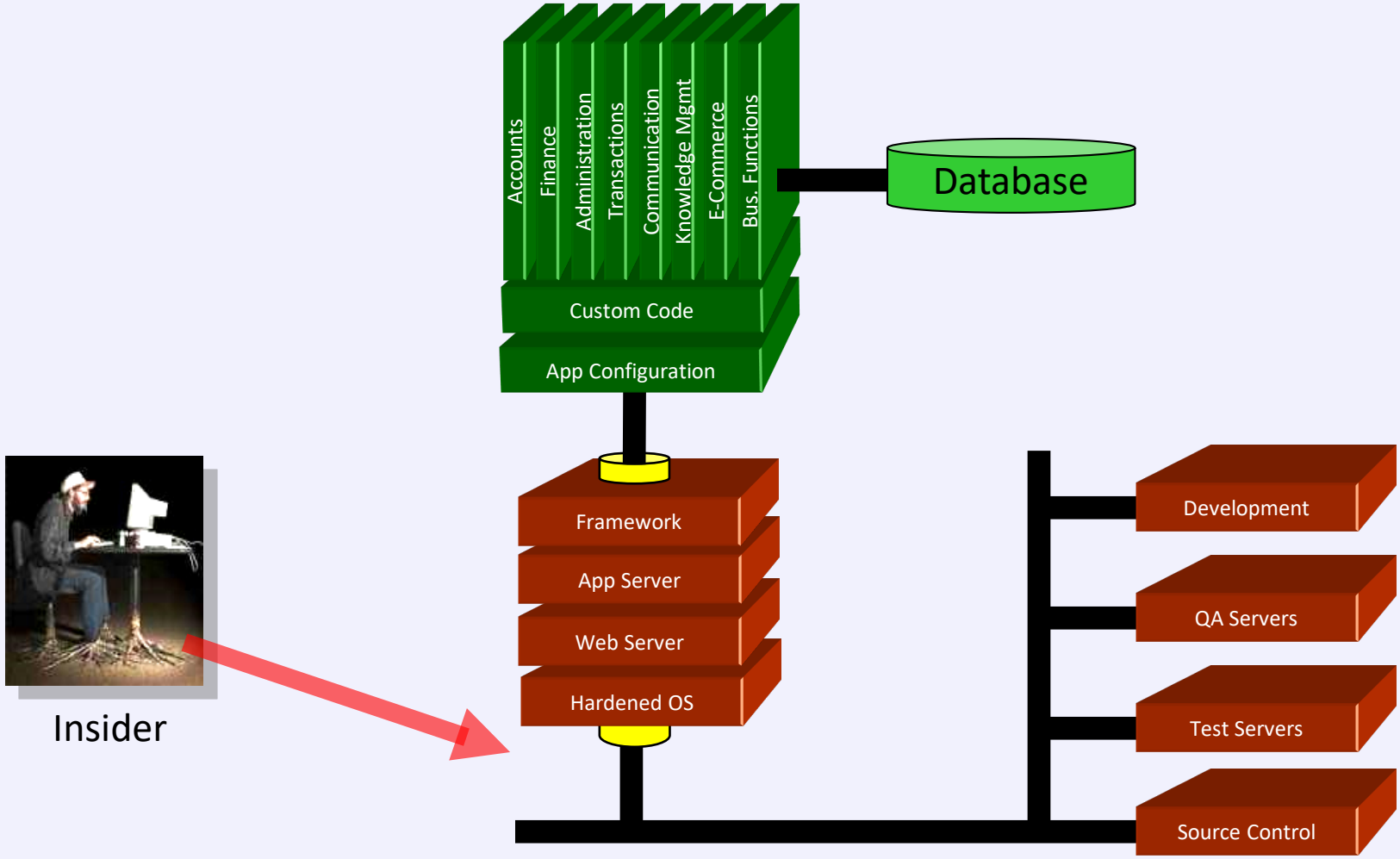
Zufällige ID	Systemspezifische Referenz
3432443	/www/image/bild.png
23467967	customerid=56
67823476879	www.target.com
43512658	…
34589345	…
6576798789	…

Backend

Mapping from 2010 to 2013 Top 10

OWASP Top 10 – 2010 (old)	OWASP Top 10 – 2013 (New)
2010-A1 – Injection	2013-A1 – Injection
2010-A2 – Cross Site Scripting (XSS)	2013-A2 – Broken Authentication and Session Management
2010-A3 – Broken Authentication and Session Management	2013-A3 – Cross Site Scripting (XSS)
2010-A4 – Insecure Direct Object References	2013-A4 – Insecure Direct Object References
2010-A5 – Cross Site Request Forgery (CSRF)	2013-A5 – Security Misconfiguration
2010-A6 – Security Misconfiguration	2013-A6 – Sensitive Data Exposure
2010-A7 – Insecure Cryptographic Storage	2013-A7 – Missing Function Level Access Control
2010-A8 – Failure to Restrict URL Access	2013-A8 – Cross-Site Request Forgery (CSRF)
2010-A9 – Insufficient Transport Layer Protection	2013-A9 – Using Known Vulnerable Components (NEW)
2010-A10 – Unvalidated Redirects and Forwards (NEW)	2013-A10 – Unvalidated Redirects and Forwards
3 Primary Changes:	▪ Merged: 2010-A7 and 2010-A9 -> 2013-A6
▪ Added New 2013-A9: Using Known Vulnerable Components	▪ 2010-A8 broadened to 2013-A7

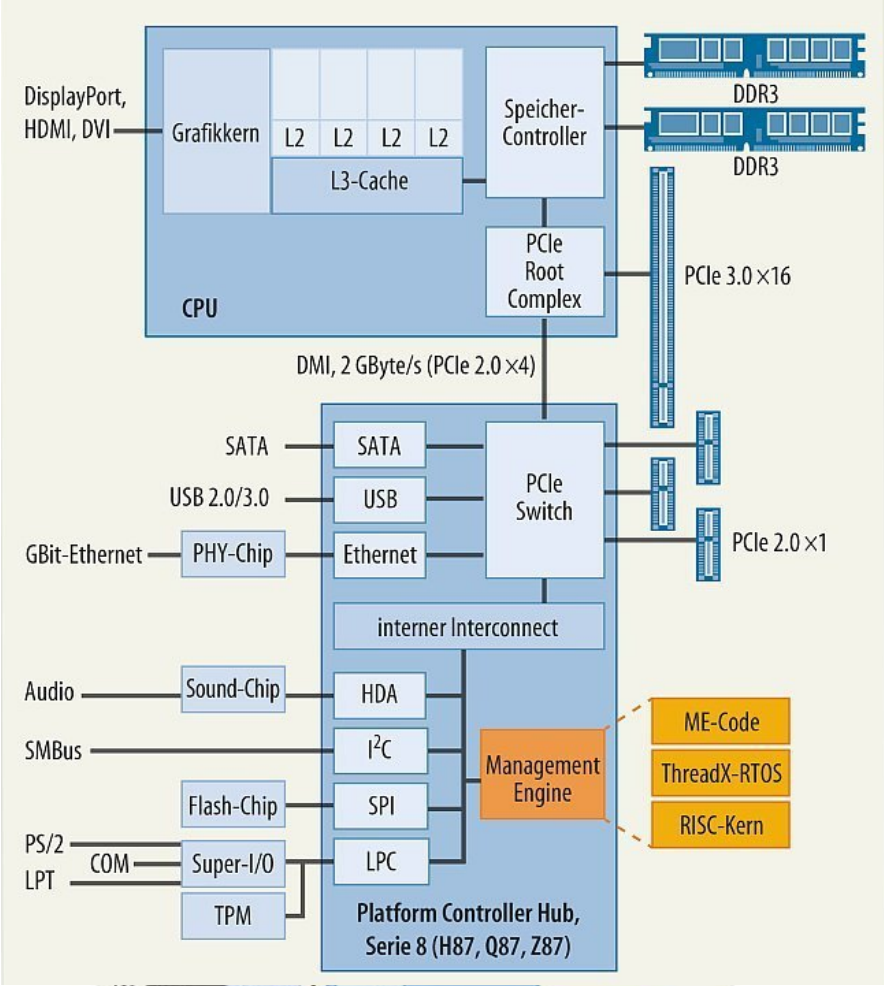
Security Misconfiguration Illustrated



Sicherheitslücke in vielen Intel-Systemen seit 2010

02.05.2017 10:54 Uhr – Christof Windeck

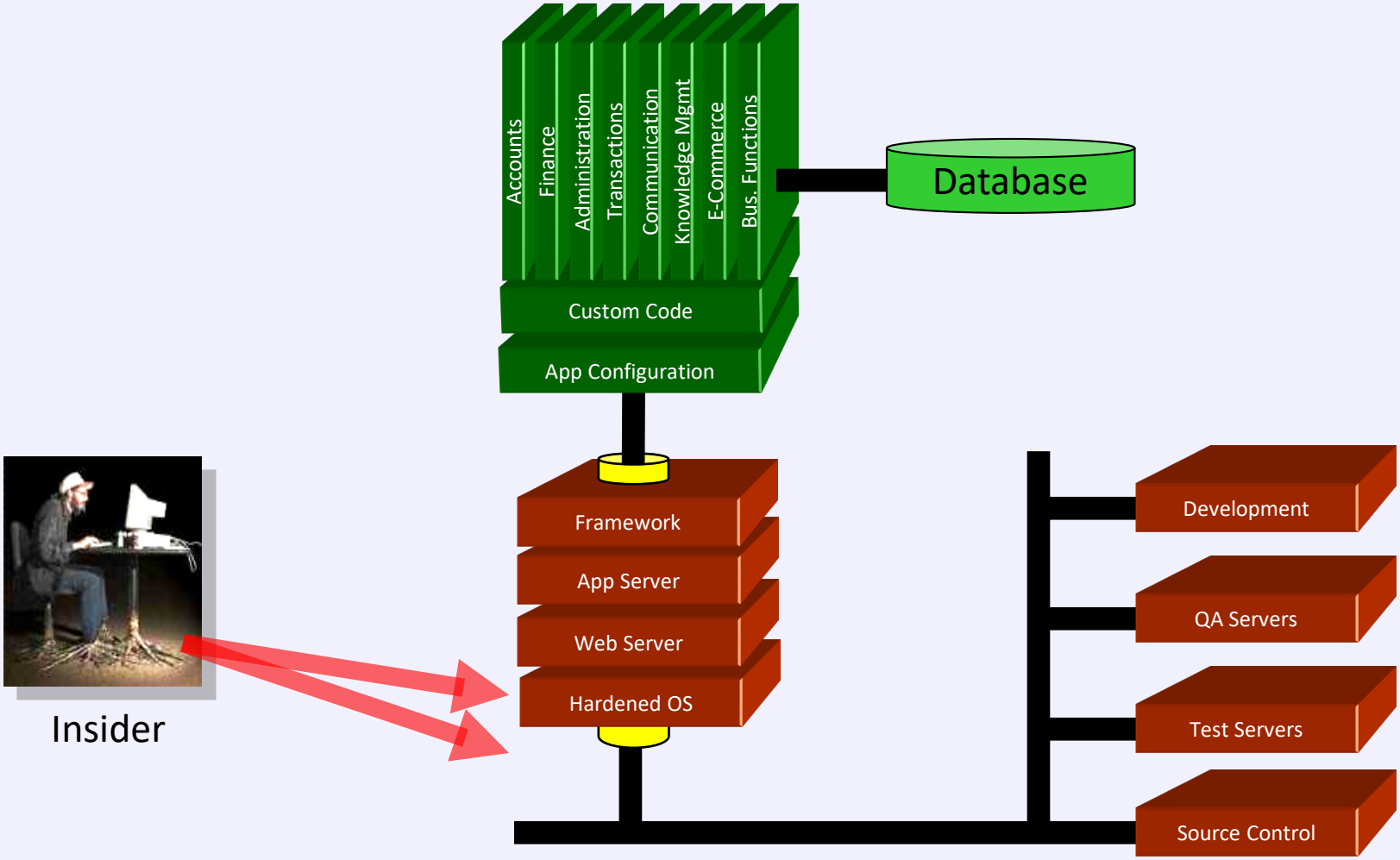
vorlesen



Die Firmware der oft kritisierten Management Engine (ME) in vielen PCs, Notebooks und Servern mit Intel-Prozessoren seit 2010 benötigt Updates, um Angriffe zu verhindern.

Security Misconfiguration Illustrated

OWASP
The Open Web Application Security Project



Dirty Cow: Linux-Rechtea

https://www.heise.de/newsticker/meldung/Dirty-Cow-Linux-Rechteausweitung-wird-fuer-Angriffe-missbraucht-3356

Dirty Cow: Linux-Rechteausweitung wird für Angriffe missbraucht

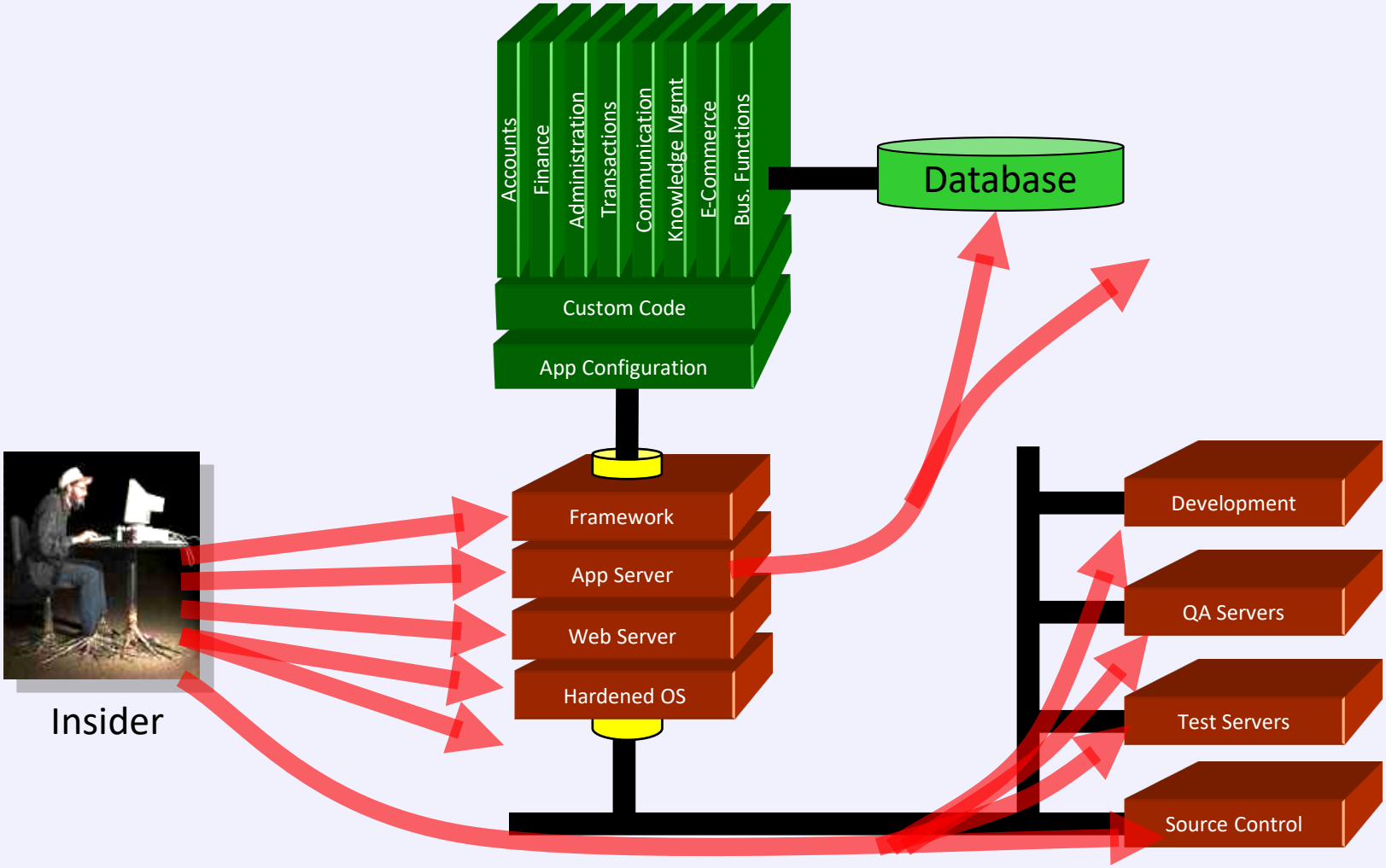
heise Security 21.10.2016 16:52 Uhr – Fabian A. Scherschel vorlesen

(Bild: dirtycow.ninja)

Die Lücke im Linux-Kernel, die Entwickler auf Grund ihrer Brisanz als "eklig" bezeichnen, wurde durch einen Angriff auf einen Webserver entdeckt. Da sie offensichtlich schon länger für Angriffe missbraucht wird, sollten Anwender jetzt schnell patchen.

Security Misconfiguration Illustrated

OWASP
The Open Web Application Security Project

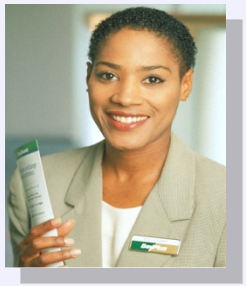


Mapping from 2010 to 2013 Top 10

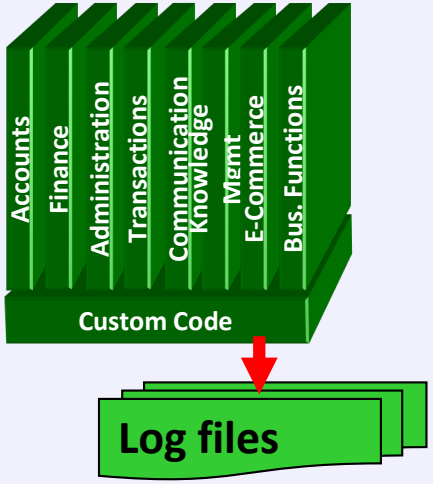
OWASP Top 10 – 2010 (old)	OWASP Top 10 – 2013 (New)
2010-A1 – Injection	2013-A1 – Injection
2010-A2 – Cross Site Scripting (XSS)	2013-A2 – Broken Authentication and Session Management
2010-A3 – Broken Authentication and Session Management	2013-A3 – Cross Site Scripting (XSS)
2010-A4 – Insecure Direct Object References	2013-A4 – Insecure Direct Object References
2010-A5 – Cross Site Request Forgery (CSRF)	2013-A5 – Security Misconfiguration
2010-A6 – Security Misconfiguration	2013-A6 – Sensitive Data Exposure
2010-A7 – Insecure Cryptographic Storage	2013-A7 – Missing Function Level Access Control
2010-A8 – Failure to Restrict URL Access	2013-A8 – Cross-Site Request Forgery (CSRF)
2010-A9 – Insufficient Transport Layer Protection	2013-A9 – Using Known Vulnerable Components (NEW)
2010-A10 – Unvalidated Redirects and Forwards (NEW)	2013-A10 – Unvalidated Redirects and Forwards
3 Primary Changes:	▪ Merged: 2010-A7 and 2010-A9 -> 2013-A6
▪ Added New 2013-A9: Using Known Vulnerable Components	▪ 2010-A8 broadened to 2013-A7

Insecure Cryptographic Storage Illustrated

OWASP
The Open Web Application Security Project

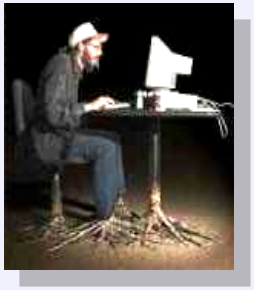


1 Victim enters credit card number in form



2 Error handler logs CC details because merchant gateway is unavailable

3 Logs are accessible to all members of IT staff for debugging purposes



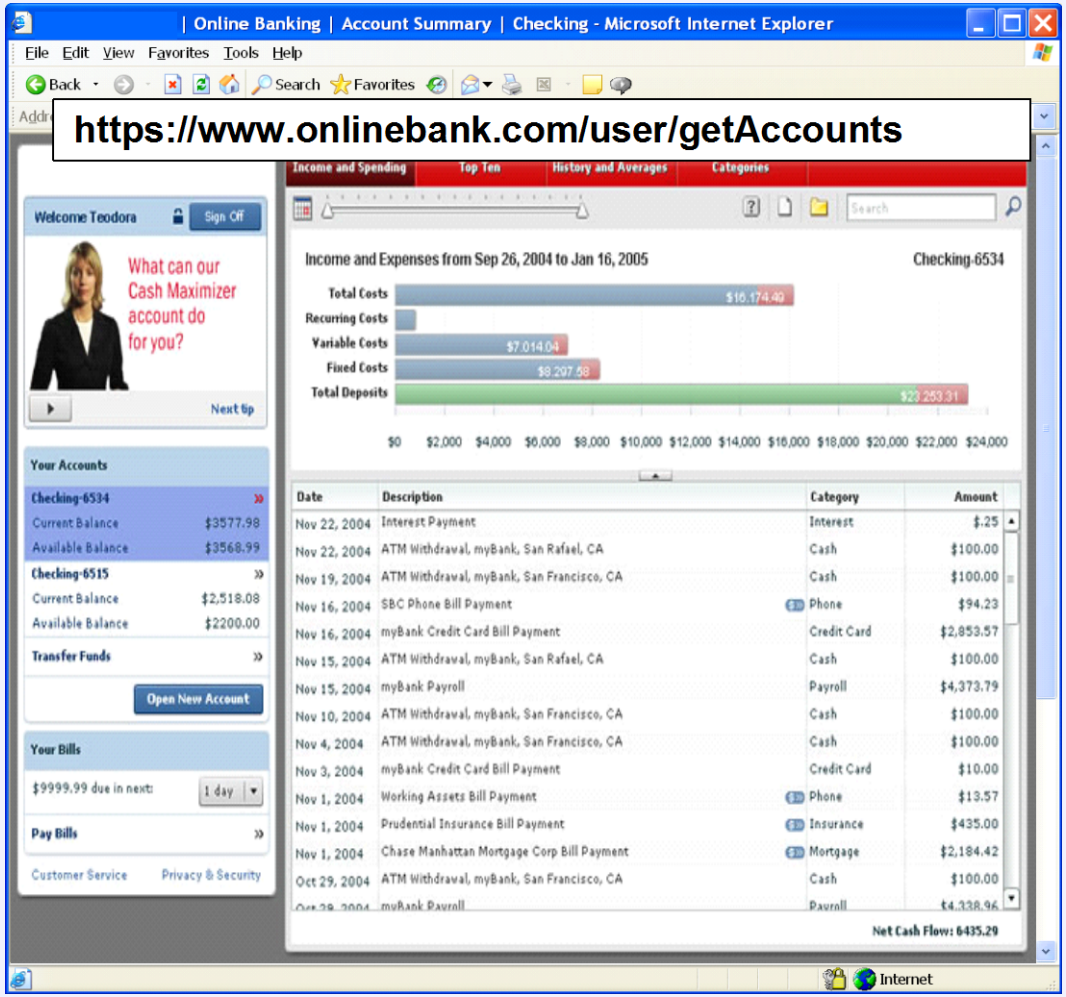
4 Malicious insider steals 4 million credit card numbers

Mapping from 2010 to 2013 Top 10

OWASP Top 10 – 2010 (old)	OWASP Top 10 – 2013 (New)
2010-A1 – Injection	2013-A1 – Injection
2010-A2 – Cross Site Scripting (XSS)	2013-A2 – Broken Authentication and Session Management
2010-A3 – Broken Authentication and Session Management	2013-A3 – Cross Site Scripting (XSS)
2010-A4 – Insecure Direct Object References	2013-A4 – Insecure Direct Object References
2010-A5 – Cross Site Request Forgery (CSRF)	2013-A5 – Security Misconfiguration
2010-A6 – Security Misconfiguration	2013-A6 – Sensitive Data Exposure
2010-A7 – Insecure Cryptographic Storage	2013-A7 – Missing Function Level Access Control
2010-A8 – Failure to Restrict URL Access	2013-A8 – Cross-Site Request Forgery (CSRF)
2010-A9 – Insufficient Transport Layer Protection	2013-A9 – Using Known Vulnerable Components (NEW)
2010-A10 – Unvalidated Redirects and Forwards (NEW)	2013-A10 – Unvalidated Redirects and Forwards
3 Primary Changes:	▪ Merged: 2010-A7 and 2010-A9 -> 2013-A6
▪ Added New 2013-A9: Using Known Vulnerable Components	▪ 2010-A8 broadened to 2013-A7

Missing Function Level Access Control Illustrated

OWASP
The Open Web Application Security Project



- Attacker notices the URL indicates his role

  /user/getAccounts

- He modifies it to another directory (role)

  /admin/getAccounts, or

  /manager/getAccounts

- Attacker views more accounts than just their own

Mapping from 2010 to 2013 Top 10

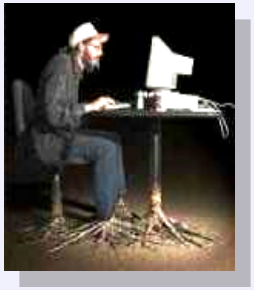
OWASP Top 10 – 2010 (old)	OWASP Top 10 – 2013 (New)
2010-A1 – Injection	2013-A1 – Injection
2010-A2 – Cross Site Scripting (XSS)	2013-A2 – Broken Authentication and Session Management
2010-A3 – Broken Authentication and Session Management	2013-A3 – Cross Site Scripting (XSS)
2010-A4 – Insecure Direct Object References	2013-A4 – Insecure Direct Object References
2010-A5 – Cross Site Request Forgery (CSRF)	2013-A5 – Security Misconfiguration
2010-A6 – Security Misconfiguration	2013-A6 – Sensitive Data Exposure
2010-A7 – Insecure Cryptographic Storage	2013-A7 – Missing Function Level Access Control
2010-A8 – Failure to Restrict URL Access	2013-A8 – Cross-Site Request Forgery (CSRF)
2010-A9 – Insufficient Transport Layer Protection	2013-A9 – Using Known Vulnerable Components (NEW)
2010-A10 – Unvalidated Redirects and Forwards (NEW)	2013-A10 – Unvalidated Redirects and Forwards
3 Primary Changes:	▪ Merged: 2010-A7 and 2010-A9 -> 2013-A6
▪ Added New 2013-A9: Using Known Vulnerable Components	▪ 2010-A8 broadened to 2013-A7

CSRF Illustrated

OWASP
The Open Web Application Security Project

1

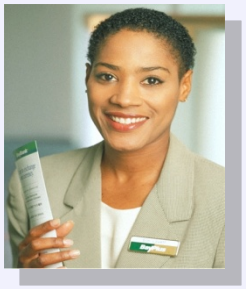
Attacker sets the trap on some website on the internet (or simply via an e-mail)

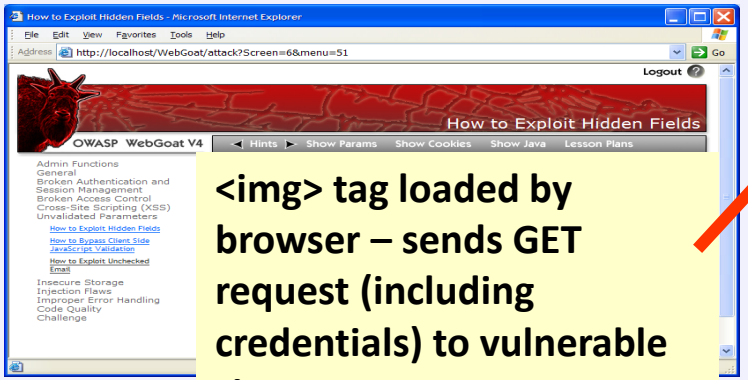


Hidden <img> tag contains attack against vulnerable site

2

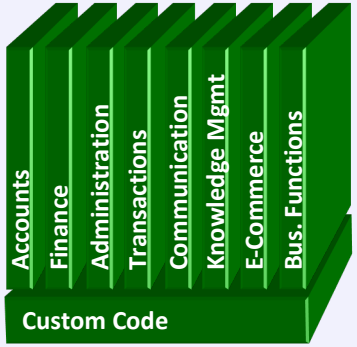
While logged into vulnerable site, victim views attacker site





<img> tag loaded by browser – sends GET request (including credentials) to vulnerable site

Application with CSRF vulnerability



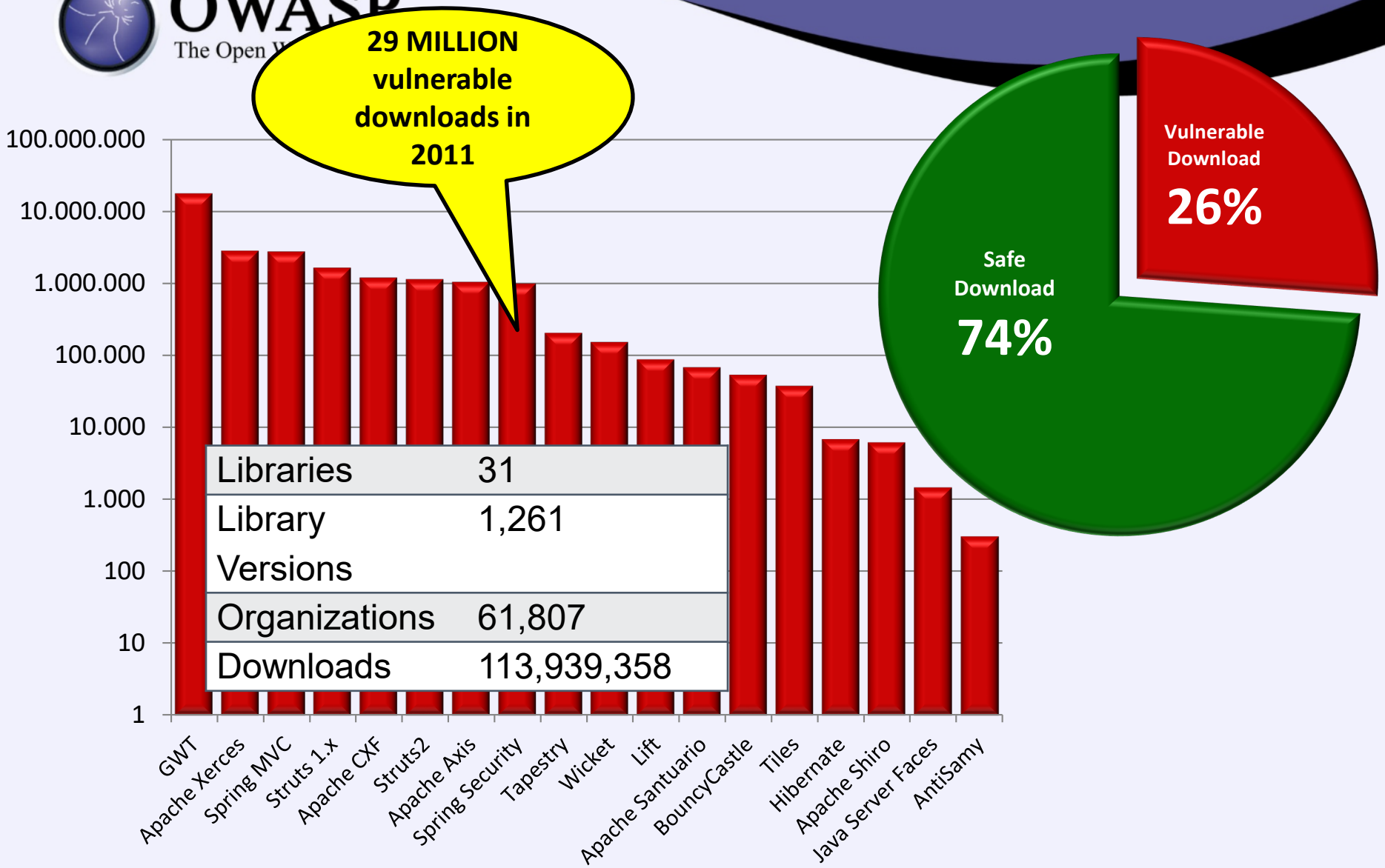
3

Vulnerable site sees legitimate request from victim and performs the action requested

Mapping from 2010 to 2013 Top 10

OWASP Top 10 – 2010 (old)	OWASP Top 10 – 2013 (New)
2010-A1 – Injection	2013-A1 – Injection
2010-A2 – Cross Site Scripting (XSS)	2013-A2 – Broken Authentication and Session Management
2010-A3 – Broken Authentication and Session Management	2013-A3 – Cross Site Scripting (XSS)
2010-A4 – Insecure Direct Object References	2013-A4 – Insecure Direct Object References
2010-A5 – Cross Site Request Forgery (CSRF)	2013-A5 – Security Misconfiguration
2010-A6 – Security Misconfiguration	2013-A6 – Sensitive Data Exposure
2010-A7 – Insecure Cryptographic Storage	2013-A7 – Missing Function Level Access Control
2010-A8 – Failure to Restrict URL Access	2013-A8 – Cross-Site Request Forgery (CSRF)
2010-A9 – Insufficient Transport Layer Protection	2013-A9 – Using Known Vulnerable Components (NEW)
2010-A10 – Unvalidated Redirects and Forwards (NEW)	2013-A10 – Unvalidated Redirects and Forwards
3 Primary Changes:	▪ Merged: 2010-A7 and 2010-A9 -> 2013-A6
▪ Added New 2013-A9: Using Known Vulnerable Components	▪ 2010-A8 broadened to 2013-A7

Everyone Uses Vulnerable Libraries



Libraries	31
Library Versions	1,261
Organizations	61,807
Downloads	113,939,358

https://www.aspectsecurity.com/news/press/the-unfortunate-reality-of-insecure-libraries

2013-A9 – Using Known Vulnerable Components

OWASP
The Open Web Application Security Project

Vulnerable Components Are Common

- Some vulnerable components (e.g., framework libraries) can be identified and exploited with automated tools
- This expands the threat agent pool beyond targeted attackers to include chaotic actors

Widespread

- Virtually every application has these issues because most development teams don't focus on ensuring their components/libraries are up to date
- In many cases, the developers don't even know all the components they are using, never mind their versions. Component dependencies make things even worse

Typical Impact

- Full range of weaknesses is possible, including injection, broken access control, XSS ...
- The impact could range from minimal to complete host takeover and data compromise

Mapping from 2010 to 2013 Top 10

OWASP Top 10 – 2010 (old)	OWASP Top 10 – 2013 (New)
2010-A1 – Injection	2013-A1 – Injection
2010-A2 – Cross Site Scripting (XSS)	2013-A2 – Broken Authentication and Session Management
2010-A3 – Broken Authentication and Session Management	2013-A3 – Cross Site Scripting (XSS)
2010-A4 – Insecure Direct Object References	2013-A4 – Insecure Direct Object References
2010-A5 – Cross Site Request Forgery (CSRF)	2013-A5 – Security Misconfiguration
2010-A6 – Security Misconfiguration	2013-A6 – Sensitive Data Exposure
2010-A7 – Insecure Cryptographic Storage	2013-A7 – Missing Function Level Access Control
2010-A8 – Failure to Restrict URL Access	2013-A8 – Cross-Site Request Forgery (CSRF)
2010-A9 – Insufficient Transport Layer Protection	2013-A9 – Using Known Vulnerable Components (NEW)
2010-A10 – Unvalidated Redirects and Forwards (NEW)	2013-A10 – Unvalidated Redirects and Forwards
3 Primary Changes:	▪ Merged: 2010-A7 and 2010-A9 -> 2013-A6
▪ Added New 2013-A9: Using Known Vulnerable Components	▪ 2010-A8 broadened to 2013-A7

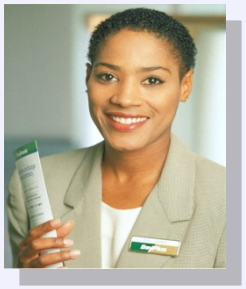
Unvalidated Redirect Illustrated

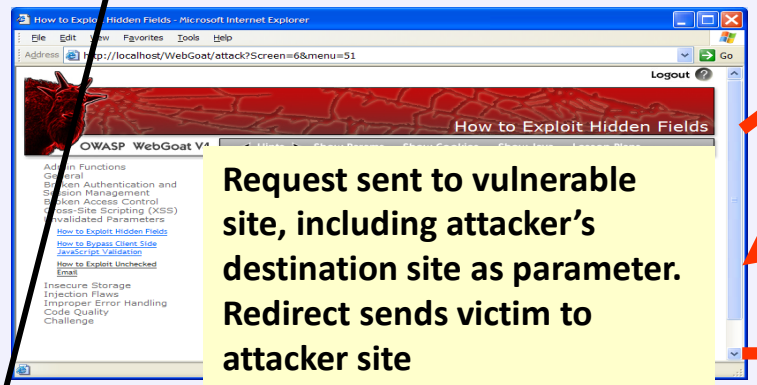
OWASP
The Open Web Application Security Project

1 Attacker sends attack to victim via email or webpage

From: Internal Revenue Service
Subject: Your Unclaimed Tax Refund
Our records show you have an unclaimed federal tax refund. Please click here to initiate your claim.

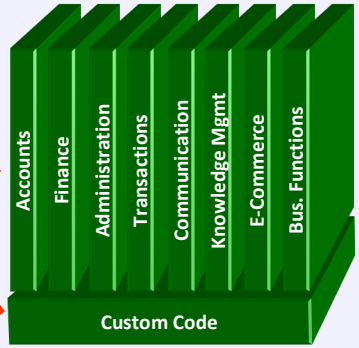
2 Victim clicks link containing unvalidated parameter

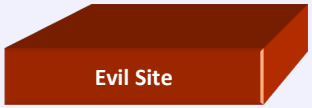




Request sent to vulnerable site, including attacker's destination site as parameter. Redirect sends victim to attacker site

3 Application redirects victim to attacker's site





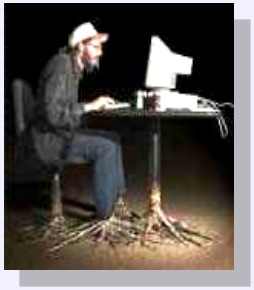
4 Evil site installs malware on victim, or phish's for private information

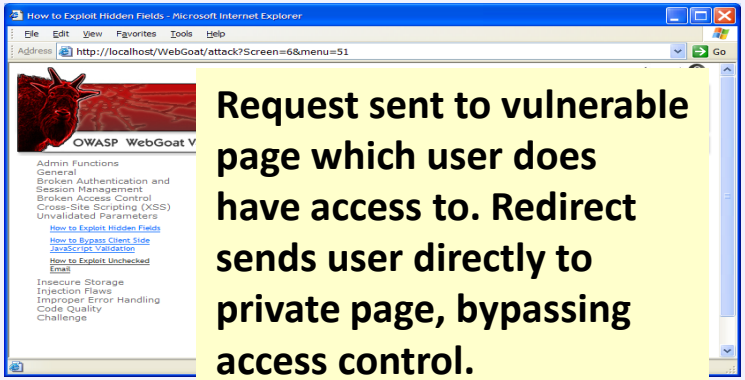
https://www.irs.gov/taxrefund/claim.jsp?year=2006
& … &dest=www.evilsite.com

Unvalidated Forward Illustrated

OWASP
The Open Web Application Security Project

1 **Attacker sends attack to vulnerable page they have access to**





Request sent to vulnerable page which user does have access to. Redirect sends user directly to private page, bypassing access control.

2 **Application authorizes request, which continues to vulnerable page**

Filter

```
public void doPost( HttpServletRequest request,
HttpServletResponse response) {
    try {
        String target = request.getParameter( "dest" ) );
        ...
        request.getRequestDispatcher( target
        ).forward(request, response);
    }
    catch ( ...
```

3 **Forwarding page fails to validate parameter, sending attacker to unauthorized page, bypassing access control**

```
public void sensitiveMethod(
HttpServletRequest request,
HttpServletResponse response) {
    try {
        // Do sensitive stuff here.
        ...
    }
    catch ( ...
```

- Liste für 2017
  - erste Version seit April
  - finale Version: November 2017

- neu aufgenommen:
  - A4: XML External Entities (XXE)
  - A8: Insecure Deserialization
  - A10: Insufficient Logging and Monitoring

Top 10 2017

OWASP
The Open Web Application Security Project

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017 - Injection
A2 – Broken Authentication and Session Management	→	A2:2017 - Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017 - Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017 - XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017 - Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017 - Security Misconfiguration
A7 – Missing Function Level Access Control [Merged+A4]	∪	A7:2017 - Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017 - Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017 - Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017 - Insufficient Logging & Monitoring [NEW, Community]

OWASP
The Open Web Application Security Project

- A4 - XML External Entities
  - Remember: XML injection attacks…
  - Denial of Service

OWASP
The Open Web Application Security Project

- A4 - XML External Entities

DoS: *One Billion Laughs*

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
    <!ENTITY lol "lollollollollollollollollollollollollollollollollollollol">
    <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
    <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
    <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
    <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
    <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
    <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
    <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
    <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
    <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
    <!ELEMENT lolz (#PCDATA)>
 ]>
<lolz>&lol9;</lolz>
```

OWASP
The Open Web Application Security Project

- A4 - XML External Entities
  - Remember: XML injection attacks…
  - Denial of Service

- Maßnahmen:
  - XML External Entities / DTD processing ausschalten
  - ggf. andere Datenformate verwenden (JSON)

OWASP
The Open Web Application Security Project

- A8 – Insecure Deserialization
  - Angriffsmöglichkeiten stark von Anwendung abhängig
  - Veränderung von serialisierten Code & Daten
    ➔ Beeinflußung der Anwendungslogik
- Maßnahmen:
  - möglichst nur einfache Datentypen verwenden
  - Serialisierte Daten & Code nur aus vertrauenswürdigen Quellen

OWASP
The Open Web Application Security Project

- A10 – Insufficient Logging & Monitoring
  - Problem: viele Angriffe werden erst nach Monaten entdeckt
- Maßnahmen:
  - Möglichst alle kritischen Systemaktivitäten loggen…
  - … und auswerten!!
  - Problem:
    - Datenschutz!
    - viele false positives

Client-seitige Sicherheitsmaßnahmen

- Client-seitige Sicherheitsmaßnahmen lassen sich leicht umgehen

- Beispiele:
  - versteckte Formularfelder
  - nicht-änderbare Formularfelder
  - Größen- / Formatbeschränkungen bzgl. Formularfeldern
  - Eingabeüberprüfungen mittels JavaScript

Same Origin Policy (SOP)

- Problem: Verhinderung von Web-Site-übergreifendem Datenaustausch
  - insbesondere durch (aktive) Inhalte
    - JavaScript, Flash, Java, …

- Lösungsidee:
  - Separierung gemäß „Herkunft" der Daten
    ➔Zugriff nur bei „same origin" erlaubt

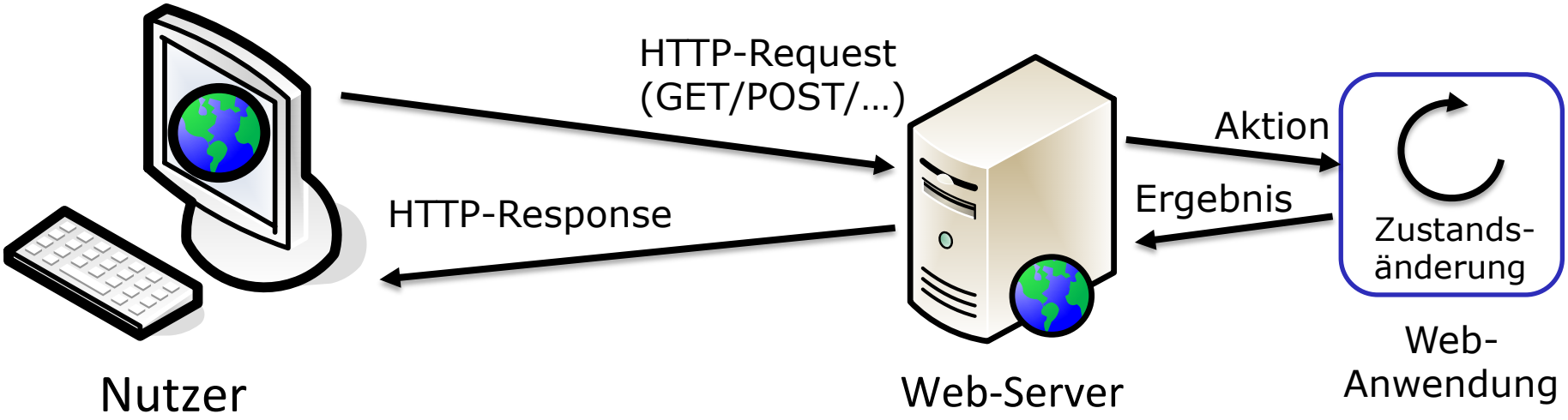
  - „same origin":
    - **Protokoll** gleich und
    - **Host** gleich und
    - **Port** gleich

Cross Site Request Forgery (CSRF / XSRF)

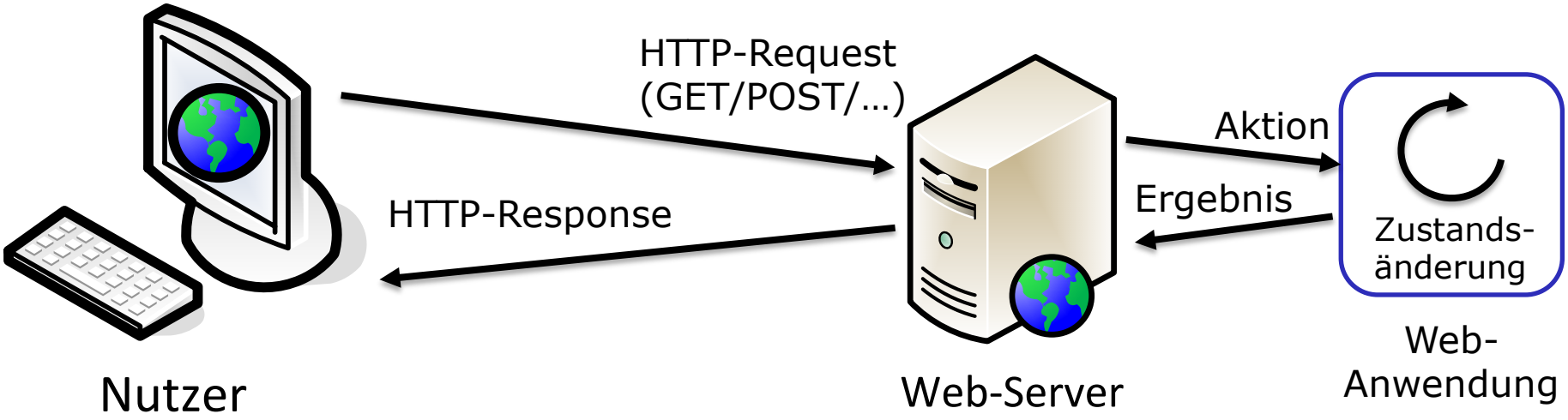
Ziel: Ausführen von **Aktionen** mit Rechten des angegriffenen Nutzers

➔ nicht notwendigerweise Informationsbeschaffung!

Ausführen von Aktionen in Web-Anwendungen



Aktionen in Web-Anwendungen – CSRF Ziele



- Aktion/Zustandsänderung ist entscheidend
  - „Schreibzugriff"
- Ergebnisinhalt/HTTP-Response von untergeordneter Bedeutung
  - Zugriff oftmals durch Same Origin Policy verhindert

Cross Site Request Forgery (CSRF / XSRF)

Ziel: Ausführen von **Aktionen** mit Rechten des angegriffenen Nutzers

➔ nicht notwendigerweise Informationsbeschaffung!

Annahme:

- Nutzer ist bei Web-Anwendung angemeldet
  - ➔ Gültige Session-ID im Browser vorhanden
    - oftmals als Cookie

Angriffsstrategie:

- Nutzer dazu bringen, unbemerkt HTTP-Request abzusetzen
  - Session-Credentials (Cookies) werden automatisch mitgesendet

Umsetzung:

- eingebettete Bilder
- automatisch generierte/abgesendete Formulare (JavaScript)
- Ausnutzung von XSS-Lücken
- …

Cross Site Request Forgery Beispiel

1. nach erfolgreichem Login setzte Web-Anwendung Cookie als Session-ID

```
<?PHP
  if(login_check())
    header("Set-Cookie: sessionid=dshf84754930jdlkf;");
?>
```

2. Web-Anwendung überprüft Cookie als Zugriffskontrolle

```
<?PHP
  if($_COOKIE["sessionid"]=="dshf84754930jdlkf")
    doAction($_GET["param");
?>
```

3. Nutzer besucht Angreifer-Webseite

```
<HTML>
  <BODY>
    <img src="http://opfer.com/?param=42">
  </BODY>
<HTML>
```

← **Session-Cookie wird automatisch gesendet**

Cross Site Request Forgery – Anmerkungen

- Angreifer kennt SessionID nicht

- Gegenmaßnahme: `REFERER`-Überprüfung
  - bietet nur bedingt Schutz
  - führt ggf. zu Benutzbarkeitsproblemen
    - Nutzerseitige Deaktivierung von `REFERER` aus Datenschutzgründen
  - Angriff: Finden von Stellen bei denen keine `REFERER`-Überprüfung möglich ist
    - Übergang HTTP → HTTPS

- Gegenmaßnahme: Shared Secret zwischen Client ←→ Server
  - wichtig: zufällig!
  - Angreifer kann gültigen Wert nicht vorhersehen
    → Konstruktion gültiger Anfragen nicht möglich
  - Beispiel: http://webapp.com/doit?param=1&**secret=34hndr2359045**
    - besser: als POST-Parameter

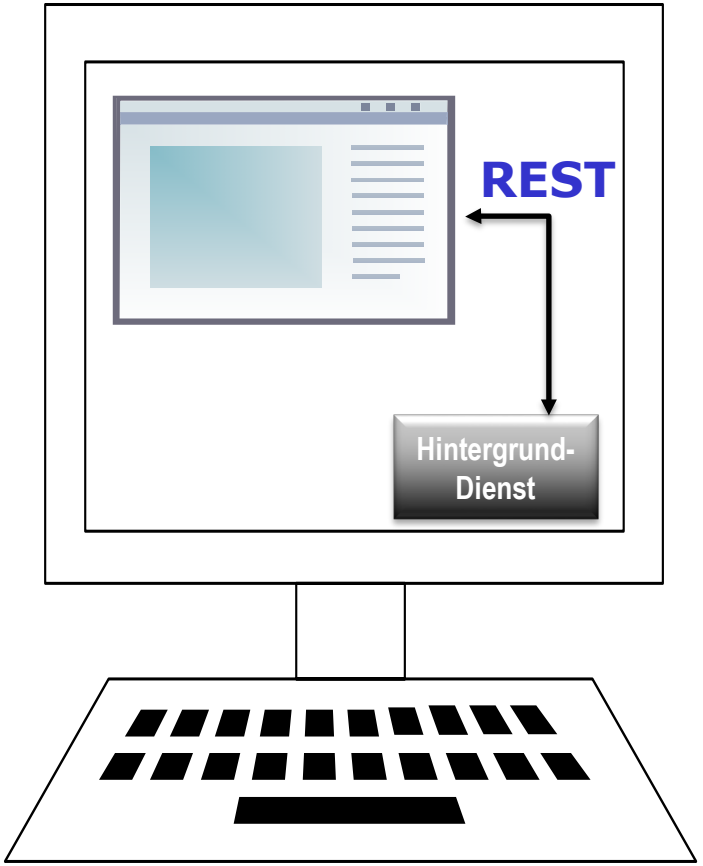
Cross Site Request Forgery – interessante Anfragen

http://fritz.box/?allowRemoteAccess=1

http://192.168.0.1:8083/fhem?cmd.Stove=set%20Stove%20on

http://localhost/gui/?action=upload&file=c:\autostart.bat

http://192.168.0.13/NAS/deleteFile?name=laptop.backup



JSON with Padding (JSONP) – Bewußtes Umgehen der Same Origin Policy

Problem: Datenzugriff über Domain-Grenzen hinweg

→ JavaScript kann Anfragen nur an „same origin" stellen

Beispiel: Web-Seite a.com:

```
<HTML>
  <BODY>
   <script>
     var xmlhttp = new XMLHttpRequest();
     xmlhttp.onreadystatechange = function() {
        var myJson = JSON.parse(this.responseText);
        doIt(myJson); }
     xmlhttp.open("GET", "http://b.com/data.json", true);
     xmlhttp.send();
   </script>
  </BODY>
<HTML>
```

Verletzung ← SOP

Lösungsidee: Ausnutzen des `<script>`-Tags zur Umgehung der SOP

- `<script>`-Tag unterliegt nicht der SOP

JSON with Padding (JSONP) – Bewußtes Umgehen der Same Origin Policy

Lösungsidee: Ausnutzen des `<script>`–Tags zur Umgehung der SOP

- `<script>`-Tag unterliegt nicht der SOP

1. Ansatz:

```
<script type="text/javascript"
        src="http://b.com/data.json">
</script>
```

➔ Abruf von http://b.com/data.json gelingt

Problem: Rückgabe `{"Foo":"bar", "id":42}` kein gültiges JavaScript

→ Fehler im Browser

2. Ansatz: JSON in gültige JavaScript-Funktion verpacken!

➔ Funktion muß bereits in Web-Seite definiert sein

- andernfalls: SOP-Problem
- Funktionsname wird Server in Parameter mitgeteilt

JSON with Padding (JSONP) – Bewußtes Umgehen der Same Origin Policy

Ansatz: JSON in gültiger JavaScript-Funktion verpacken!

➔ Funktion muß bereits in Web-Seite definiert sein (sonst: SOP-Problem)

```
<script>
  myFunction(strJson)
    {
      doIt(JSON.parse(strJson));
    }
</script>
```

- Funktionsname wird Server in Parameter mitgeteilt

```
<script type="text/javascript"
        src="http://b.com/data.json?callback=myFunction">
</script>
```

➔ Payload vom Server für http://b.com/data.json?callback=myFunction

```
myFunction({"Foo":"bar", "id":42});
```

JSONP – Sicherheitsprobleme

- Third-Party Server muß vertraut werden
  ➔ insbesondere bezüglich Injection-Angriffen

Beispiel: Payload von http://b.com/data.json?callback=myFunction

```
myFunction({"Foo":""});malliciousJavaScript();//", "id":42});
```

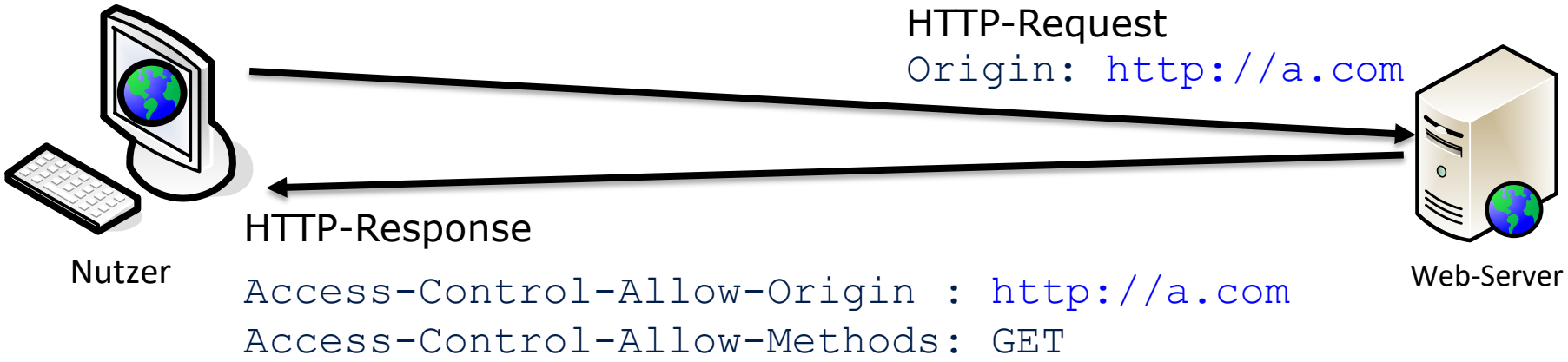
- Cross-Site Request Forgery ist leicht möglich
  - problematisch wenn Rückgabe sensible Informationen enthält

```
<script>
     myFunction(strJson){ stealCredentials(strJson)); }
</script>
<script type="text/javascript"
 src="http://opfer.com/credentials.json?callback=myFunction">
</script>
```

www.attacker.com

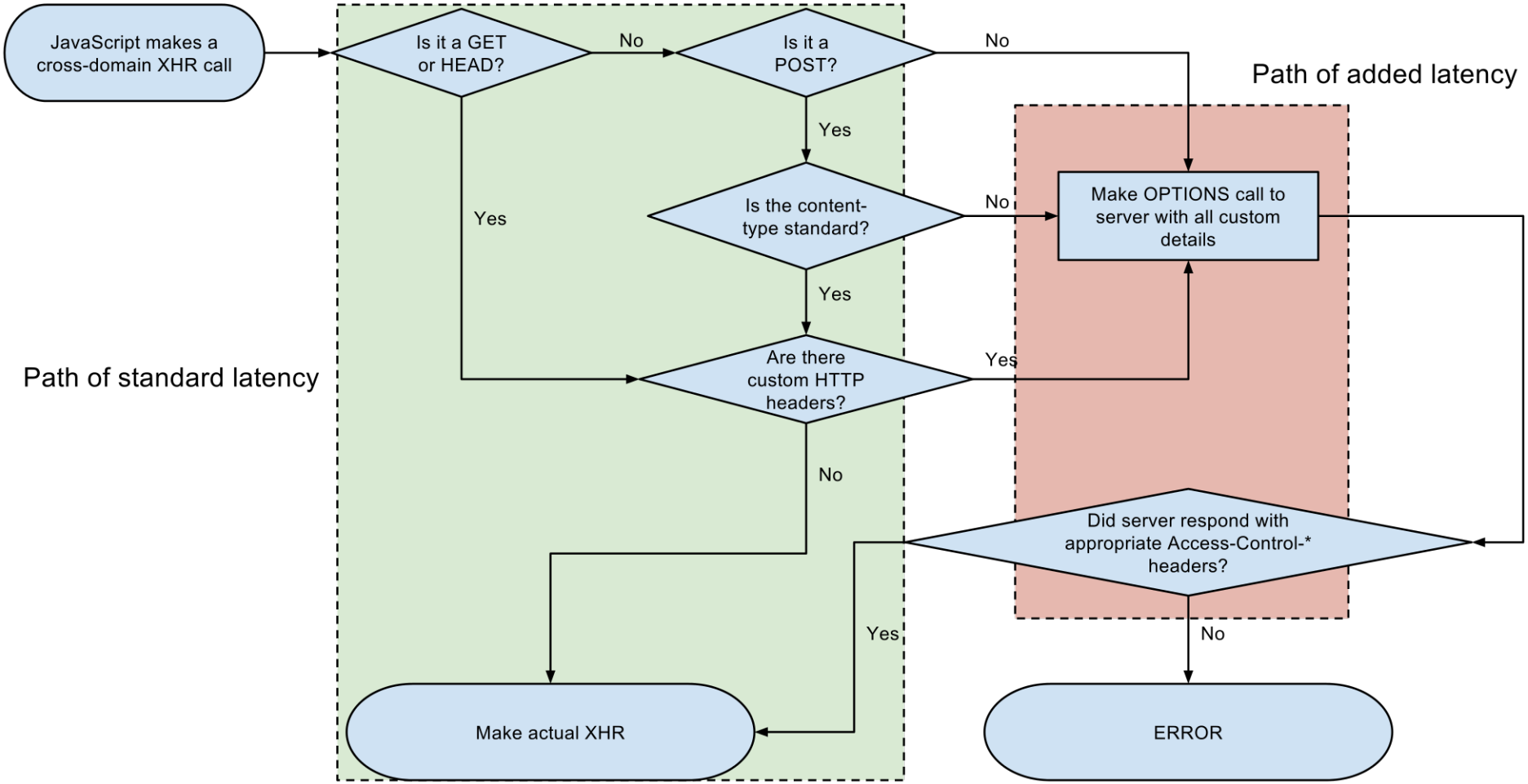
Cross Origin Resource Sharing – CORS

- Problem: Same Origin Policy verhindert ***gewünschten*** Origin-übergreifenden Zugriff
  - insbesondere bei komplexen modernen Web-Anwendungen nötig
- Lösungsidee:
  - Server teilt mit, welche anderen Origins auf die ausgelieferten Inhalte zugreifen dürfen
    - White-Listing-Ansatz
- Umsetzung: CORS
  - spezielle HTTP-Header



CORS – Pre-flight Requests

- vom Browser vor eigentlicher Anfrage gesendet
  - Überprüfung der Zugriffsregeln / Kompatibilität



CORS – Sicherheitsprobleme

- Zugriffskontrolle nicht restriktiv genug
  - `Access-Control-Allow-Origin: *`

Session Fixation Attacks

Ziel:

- Ausführen von Aktionen im Namen angemeldeter Nutzer

Strategie bisher:

- unberechtigte Kenntnisnahme von SessionID

neue Strategie:

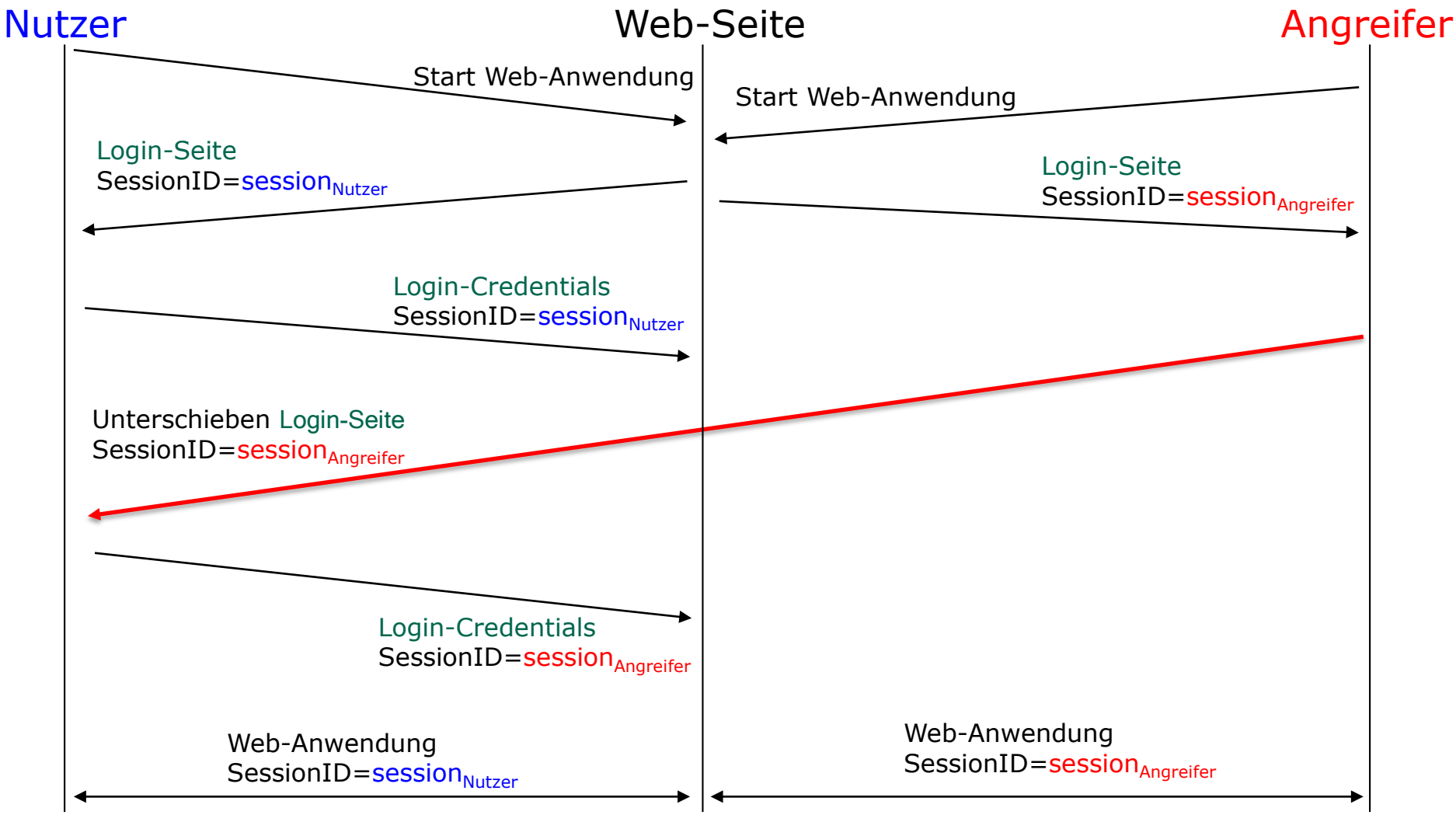
- SessionID durch Angreifer festgelegt
  ➔ *Session Fixation*

Umsetzung:

- Ausnutzen von unsicheren Web-Anwendungen, bei denen **`SessionID`** vor und nach Authentikation gleich ist

Session Fixation Attacks

Problem: SessionID vor und nach Authentikation gleich



Session Fixation Attack – Durchführung

1. Session Setup
   - „echte" Session mit Ziel-Server etablieren, um SessionID zu erhalten
     - „striktes" Session-Management
   - selbstgewählte SessionID benutzen
     - „permissive" Session-Management

2. Session Fixation
   - SessionID dem Nutzer unterschieben
     - Nutzer-SessionID wird „fixiert" auf Angreifer-SessionID

3. Session Entrance
   - Warten bis Nutzer SessionID „gültig macht"

Session Fixation Attack – Session Fixation Durchführung

- SessionID als URL Parameter
  - Angreifer sendet präpariert URL an Nutzer
    - Phising
    - URL-Verkürzungsdienst
- SessionID in verstecktem Formular-Feld
  - gefälschte Web-Seite mit präpariertem Login-Formular
    - Hm, Angreifer könnte gleich Nutzername/Paßwort stehlen…
  - XSS-Schwachstelle ausnutzen
    - dito

Session Fixation Attack – Session Fixation Durchführung

- SessionID im Cookie gespeichert
  - XSS-Schwachstelle ausnutzen
    - Angreifer könnte vermutlich auch SessionID stehlen
  - Verbesserte Version: Domain-Cookie setzen
    - XSS-angreifbarer Server: www.a.com

      ➔Domain-Cookie setzen:

      ```
      http://www.a.com/?<script>
      document.cookie=„sessionid=1234;domain=.a.com"
      </script>
      ```

    - Nutzer geht auf: onlinebanking.a.com

      ➔ Domain-Cookie wird automatisch gesendet

Session Fixation Attack – Session Fixation Durchführung

- SessionID im Cookie gespeichert
  - `<META>`-Tag
    - Injection-Angriff
    - statt JavaScript → `<meta http-equiv="Set-Cookie" content="sessionid=1234">`
      - funktioniert auch außerhalb von `<HEADER></HEADER>`
      - ggf. nicht gefiltert von Web-Anwendung
  - HTTP-Antwort-Header
    - Manipulation des Netz-Verkehrs
      - Hm, hier schienen bessere Angriffe möglich…
    - Kontrolle über Server in Domain
      - Domain-Cookie setzen
      - DNS-Angriff → Einträge fälschen

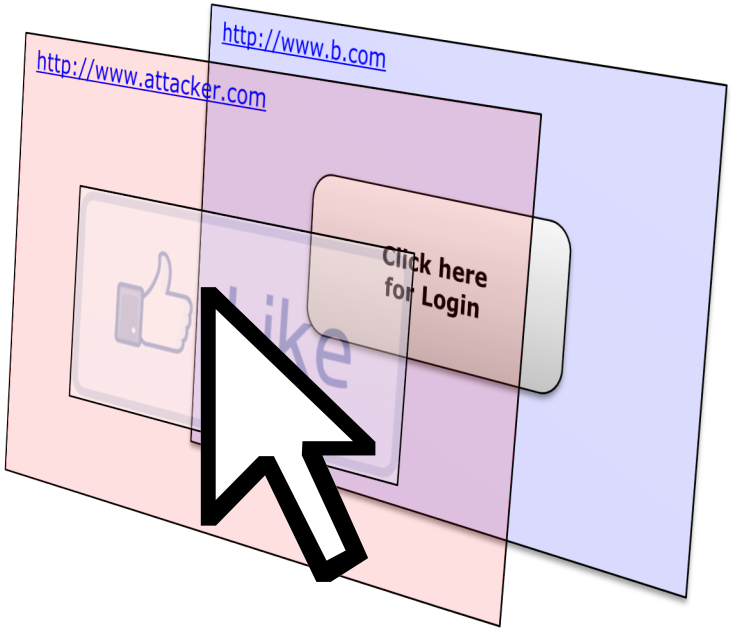
Clickjacking

Ziel: Aktivitäten im Browser durch Nutzer auszulösen

Lösungsidee: Nutzer soll Aktionen unbemerkt auslösen

Umsetzung:

- Überlagern von Web-Inhalten mit unsichtbaren Angreifer-Inhalten
  - ➔ Nutzer denkt, Aktivitäten auf **sichtbaren** Inhalten durchzuführen – tatsächlich werden diese aber auf den **unsichtbaren** durchgeführt



Clickjacking – Beispiel

```
<html><head><style>iframe {
width: 900px; height: 200px;
// Absolute Positionierung, um Button über Link zu legen
position: absolute;
top: 50px; left: -650px; z-index: 2;
// Unsichtbar machen
opacity: 0.0; filter: alpha(opacity=0.0); }
button {
position: absolute;
top: 50px; left: 100px; z-index: 1; width: 120px; }
</style></head>
<body>
<iframe src="http://www.slub-dresden.de"
   scrolling="no"></iframe>
<button>Click Here!</button>
</body></html>
```

← **iframe für unsichtbaren Inhalt**

← **iframe ist im Vordergrund**

← **aber unsichtbar**

← **CSRF!**

← **sichtbar, unter unsichtbarem Ziel platziert**

Cross-Site-Image-Overlaying (XSIO)

- Ziel:
  - Nutzer auf Angreifer-Web-Seite locken
  - CSRF
- Lösungsidee:
  - Angreifer-Elemente auf vertrauenswürdiger Opfer-Web-Seite platzieren
- Umsetzung:
  - Injection-Angriff → diesmal: `<img>`-Tag (+ Verweis `<a>`)
- Konkret: absolute Positionierung verwenden, um Teile der Web-Seite zu verändern

```
<a href="http://evil.com">
  <img src="http://evil.com/amazonlogo.png"
     style="position:absolute;right:30px;top:100px;"/>
</a>
```

Drag-and-Drop Hijacking

- Ziel: Umgehen der Same Origin Policy
- Lösungsidee:
  - Ausnutzen des HTML5 Drag-and-Drop-API
  - Drag-and-Drop unterliegt nicht der SOP
- Umsetzung:
  - ähnlich Clickjacking
  - Hintergrund:
    - harmlose Drag-and-Drop-Anwendung (Spiel)
  - Vordergrund:
    - unsichtbar
    - enthält:
      - Opfer-Elemente, die zu stehlenden Inhalt enthalten
      - Angreifer-Elemente, in die der Inhalt abgelegt wird

File Upload Vulnerabilities

- Ziel
  - Ausführen von Befehlen auf Web-Server

- Lösungsidee:
  - File-Upload-Funktion ausnutzen

- Umsetzung:
  - Befehle in hochgeladenen Dateien „verstecken"
  - Ausführen durch Web- / URL-Aufruf
  - komplette „Web-Shell" in einer Datei

- Problem:
  - hochgeladene Datei werden gefiltert
    - basierend auf Dateiendung
      ➔Umgehen mittels Groß-/Kleinschreibung, Kodierung („.PhP")
      ➔‚0' einfügen: `attack.php[0x00].jpg` → Datei: `attack.php`
    - basierend auf MIME-Type
      ➔ MIME-Type fälschen

File Upload Vulnerabilities

- Problem:
  - hochgeladene Datei werden gefiltert
    - ➔ Ausführungsumgebung anpassen: harmlose Datei → ausführbare Datei
    - ➔ .htaccess hochladen!

```
<FilesMatch „*.php.gif“>
   SetHandler application/x-httpd-php
 </FilesMatch>
 AddType application/x-httpd-php .phq
```

File Upload Vulnerabilities

- Problem:
  - Filterung basierend auf „gültigem Inhalt"

```
<?php>
  if(!getimagesize($uploadedFile))
    exit(-1);
  moveFile($uploadedFile,"pictures/".$uploadedFile);
?>
```

← **Bild?**

➔ Umgehung: gültiges Bild mit eingebetteten Befehlen

exiftool –comment  = <?php phpinfo ?>

Gegenmaßnahmen

- OWASP Cheat Sheets!
  - https://www.owasp.org/index.php/Cheat_Sheets
  - enthält Checklisten bezüglich potentieller Schwachstellen und Gegenmaßnahmen
    - gegliedert nach den verschiedenen Technologien
- Bibliotheken / Frameworks benutzen