Web Security

Prof. Dr.-Ing. Andreas Heil

v1.0.0

Lernziele

Sie lernen, dass alles, wirklich alles furchtbar schrecklich ist...

Ursprünge

- Anfangs wurde von Web Servern vorrangig statischer Content ausgeliefert
- Anbindung and Datenbanken
- Security auf Server-Seite bezogen
  - Buffer Overflow Attacks
  - Erlangen privilegierter Rechte im Web Server selbst
  - SQL Injection

Heute

- Browser sind kompliziert
  - JavaScript - client-seitige Skripte
  - DOM - Verändern der Dokumente via JavaScript
  - XML/HTTP Requests - Asynchrone Kommunikation mit dem Server (AJAX)
  - Web Sockets - Erlauben Full-Duplex-Kommunikation
  - Multimedia
  - Geolocation (Smartphone, Desktop via WiFi!!!)
  - Google Native Client > WebAssembly
  - ...

Komplexität

- Verschiedene Sprachen
- Verschieden Plattformen
- Verschiedene Chipsätze

Parsing Context

Folgendes Tag in einer Webseite

```
<script>var x = "untrusted source code...</code>
```

Welche Probleme können bei diesem Code-Beispiel entstehen?

Noch ein Beispiel

```
<body></script>...
```

Problem mit Web Standards

- Web Standards sind lange
- Web Standards sind langweilig
- Web Standard sind manchmal widersprüchlich
    - quirksmode.org

Beispiel Seite

http://foo.com

- Werbung als GIF von ads.com
- Analytics Library von g.com
- jQuery von cdn.foo.com
- Web Fonts von fonts.g.com
- HTML
  - Inline JS von foo.com
  - Frame https://fbook.com/likethis.aspx
    - Inline JS fron https://fbook.com
    - j.jpeg von https://fbook.com

Einige Fragen

- Kann der Analytics-Code auf den jQuery-Code zugreifen?
- Kann der Analytics-Code z.B. mit Input-Elemente im HML interagieren?
- Was sit die Beziehung zwischen dem fbook-Frame und dem Rest der Seite?
  - fbook ist https, foo.com ist http
  - Wie können beide Seiten miteinander interagieren?

Sicherheitsmodell in Browsern

- Same-origin Policy
  - Ziel: Zwei Web Seiten sollten nicht miteinander kombiniert werden können
  - War früher einfach, heute fast unmöglich
- Strategie
  - Jede Resource wird einer Quelle (engl. Origin) zugeordnet
  - JS kann nur auf Ressourcen zugreifen, die von der gleichen Quelle stammen
  - Definition Origin: `scheme + host + port`
  - Beispiele:
    - http://foo.com/index.html
    - https://foo.com/index.html
    - http://bar.org:8181/buzz.php

Vier Konzepte

1. Jeder Origin hat client-seitige Ressourcen

- Cookies
- DOM Storage
- JavaScript Namespace
- DOM Baumstruktur

2. Jedes Frame hat die Quelle seiner URL
3. Skripte werden mit der "Authority" Ihres Frames ausgeführt
4. Passiver Inhalt erhält keine "Authority"

- Bilder
- CSS Dateien

Interaktion zwischen Frames

- `postMessage()`[^1]
  - Beide Seiten müssen zustimmen
  - Frames von anderer Herkunft können keine HTTP-Requests auf das übergeordnete Frame ausführen

MIME Sniffing Attack

Angriffsvektor bzgl. Punkt 4

- Alte Browser z.B. IE
- Dateien (statischer Content) auf Basis der ersten 256 Bytes interpretiert
- Ermöglicht HTML Code mit JavaScript einzuschleusen, der dann im Kontext der aufrufenden Seite ausgeführt wird
- Aus Seite des Opfers sieht der Content der Angreifers wie statischer Content aus
- IE interpretiert den Content allerdings als HTML und JavaScript
- Statischer Content hätte keine Authority, das Skript jedoch schon, da es im Kontext der übergeordneten Seite ausgeführt wird

Frames & Window Objekte

- Frames - eigenständige JS-Universen
- Windows - Alias für den globalen Namensraum

Origin: Frame URL ODER Suffix der urspünglichen URL via 'domain(document.domain

- x.y.z.com
- y.z.com
- z.com
- a.y.z.com
- .com

Was macht Sinn, was nicht?

Frames und Same Origin Policy

Zwei Frames können aufeinander zugreifen

- wenn beides document.domain auf den gleichen Wert setzen
- wenn keines der frames document.domain geändert wurden

Warum? Man möchte verhindern, dass eine Seite von einer bösartigen Sub-Domain angegriffen wird.

- DOM Nodes erhalten die Origin des Frames
- Cookies: Domain + Pfad
  - `*.hs-heilbronn.de/aheil`
  - `docment.cookie` (via JS auf Client-Seite)
  - Server-Seitig via HTTP-Repsonse

Was ist das Problem mit Cookies

- Client-Seite kann "Per-user" Daten Speichern
  - keine gute Idee dass jeder dort reinschreiben kann
  - Gängiger Angriffsvektor
- Beispiel: `foo.co.uk`
  - Was ist mit `co.uk`?
  - https://publicsuffix.org/learn/ -> Regeln und Bibliotheken für Domain-

- X-Origin Request
  - Same Origin, außer CORS ist im Response-Header erlaubt
  - Header: `Access-Control-Allow-Origin` foo.com

Bilder, CSS und JS

- Können prinzipiell von überall her geladen werden
  - Frame kann sich die Bits des Bilds nicht anschauen
- CSS von überall
  - Text kann nicht direkt angeschaut werden
- JS
  - Fremder JS-Code kann ausgeführt werden
  - Fremder Code kann aber nicht mit inspect "angeschaut" werden

Cross-Site Request Forgery (CSRF)

- Bei HTTP-Request, werden die betreffenden Cookies mitgesendet
- URL kann analysiert werden und Attacker kann diese ausführen
- `https://meinebank.de/transfer?wert=1000&an=hacker`
- Vermeidung durch Verwendung einen Nounce `...=hacker&csfr=ACE34F21`
- Cheatsheet[^2]

Netzwerkadressen

- DNS Rebinding Address
  - Ziel: JS des Angreifers ausführen
    - Domain Name registrieren: hacker.com + setzt DNS Server auf
    - User besucht hacker.com
    - Browser generiert ein DNS Request für hacker.com
    - Angreifer Antwort: sehr kurze TTL
    - Angreifer: Ändert DNS Server hacker.com auf die IP des Opfers
    - Web Seite holt jetzt z.B. etwas via AJAX landet auf der Seite des Opfers
    - Code beim Client kommt vom Angreifer, kann aber auf eine andere Quelle (Origin) zugreifen

Pixel

- Click Jacking Attack
  - Überlagernde Frames
  - Via JS prüfen ob man in einem Child Frame läuft `if (self!=top)`
  - HTTP-Response Header: `X-Frame-Options`, dass der Content nicht in ein Frame gepackt werden darf

More to be come...

Referenzen

Acknolegments

Diese Vorlesung basiert auf der Security Vorlesung von Prof. James Mickensen, MIT.