

Web-basierte Systeme

o8: Architektur moderner Browser

Wintersemester 2024

Rüdiger Kapitza



Lehrstuhl für Verteilte Systeme
und Betriebssysteme



Friedrich-Alexander-Universität
Technische Fakultät

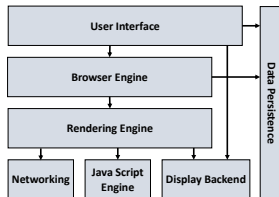
Architektur moderner Browser

Vorlesungsplan

- 18. Oktober** Einführung und Darstellung von Webseiten (HTML und CSS)
- 8. November** Hypertext Transfer Protocol
- 15. November** Browser Schnittstellen
- 22. November** Kommunikationsschnittstellen im Browser
- 6. Dezember** WebAssembly
- 20. Dezember** **Architektur moderner Browser** und Vorbereitung Papieranalyse
- 10. Januar** Clientseitige Architekturmuster und serverseitige Implementierung von Web-basierten Systemen
- 17. Januar** Papieranalyse
- 24. Januar** Caching bzw. Lastverteilung durch Zwischenspeicher
- 31. Januar** Aspekte von Web Sicherheit
- 7. Februar** Zusammenfassung und Ausblick

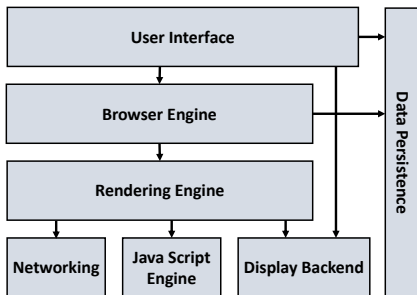
Zielsetzung der Lerneinheit

- Allgemeines Verständnis für die Architektur eines Browsers
- Verständnis für Fusion von HTML, CSS und JS zu einem Layout
- Kennenlernen von Systemansätzen zur Verbesserung der Leistungsfähigkeit & Sicherheit von Browsern
 - Beispiel: Chromium



Generische Softwarearchitektur eines Browsers

- Vgl. *A Reference Architecture for Web Browsers* (von 2005!) [2]
- Arbeit stellt eine generische Softwarearchitektur vor welche durch Analyse verschiedener Browser erarbeitet wurde



User Interface

- Schicht zwischen Nutzer und der Engine des Browsers
- Stellt die Oberfläche bereit: Toolbar, Rahmen mit Fortschrittsanzeige, Masken für Einstellungen, usw.
- Nicht selten Einbettung in den Desktopmanager des Systems

Browser Engine

- Implementiert Basisfunktionen des Browsers und stellt eine Schnittstelle zur Rendering Engine bereit
- Basisfunktionen sind bspw.: Laden einer URI, Navigieren im Verlauf der bisher aufgesuchten Seiten oder das erneute Laden einer Seite
- Schnittstelle zum Abfragen weiterer Informationen einer Sitzung
 - Bspw.: Fortschritt bzgl. der aktuell zu ladenden Seite oder Warnungen der JavaScript Engines
- Schnittstelle um den Zustand der Rendering Engine abzufragen, sowie Einstellungen an diesem vorzunehmen

Rendering Engine

- Zuständig für die Darstellung einer HTML Seite unter Anwendung von CSS und der Integration von Grafiken
- Es wird das genaue Layout festgelegt (Positionierung aller Elemente) und dieses erneuert, wenn weitere Informationen vorliegen
- Zentrale Komponente der Rendering Engine sind HTML- und CSS-Parser

Networking

- Unterstützung für den Transfer von Daten via HTTP(S) (und FTP)
- Konvertierung zwischen Zeichensätzen, sowie Unterstützung für MIME media Typen
- Evtl. Unterstützung für die Zwischenlagerung aktueller Daten

JavaScript Interpreter

- Unterstützung für JavaScript (ECMAScript)
- Interaktion mit der Rendering Engine

XML Parser

- Ursprünglich ein separates Architekturelement heute wohl subsumiert in anderen Komponenten

Display Backend

- Stellt Grafikprimitive bereit
- Schnittstellen zum Desktopmanager und dem Betriebssystem
 - Bspw.: Look & Feel unter Linux vs. Windows und Schrifttypen

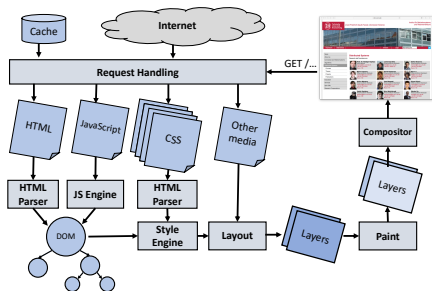
Data Persistence

- Verwaltung der Daten während einer Session
- Andere Verwaltungsdaten: Lesezeichen, Einstellung der Oberfläche
- Zwischenspeicher (Cache)
- Sicherheitsinformationen: Zertifikate etc.

Architekturüberblick (Firefox Quantum)

Architekturüberblick mit Fokus auf Firefox Quantum

- Entwicklerblog welcher die Funktionsweise von Quantum beschreibt ¹
- *Achtung*: Artikel diskutiert die *Browser Engine* – im Sinne von [2] entspricht dies aber *eher* der Rendering Engine



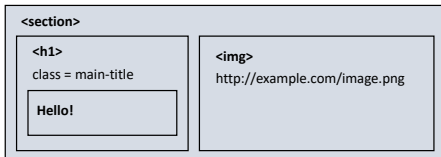
¹<https://hacks.mozilla.org/2017/05/quantum-up-close-what-is-a-browser-engine>

Vereinfachter Ablauf: Laden und Parsen

- Daten werden durch das Request Handling-Modul geladen
 - HTML, Grafiken, CSS und JavaScript
- HTML-Parser wandelt HTML-Dokumente in einen Document Object Model (DOM)-Knoten – einem sogenannten Inhaltsbaum um
- Andere im HTML enthaltene Style-Daten und Skripte werden ebenfalls geladen und weiterverarbeitet
- Skripte können im Prinzip die Seitenstruktur und Style-Daten verändern bevor ein HTML-Dokument vollständig geparkt wurde

Vereinfachter Ablauf: Laden und Parsen

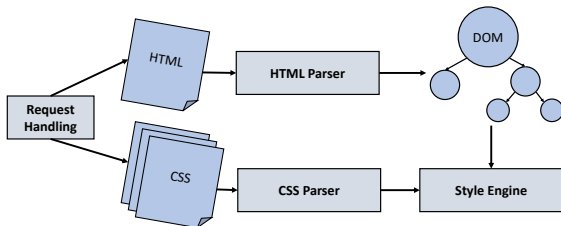
```
1 <section>
2   <h1 class="main-title">Hello!</h1>
3   
4 </section>
```



Architekturüberblick (Firefox Quantum)

Vereinfachter Ablauf: Verarbeitung von CSS

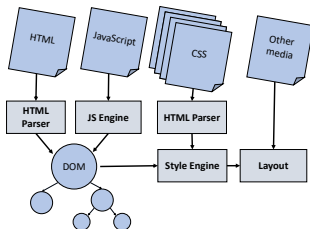
- Style-Anweisungen werden ebenfalls geparkt und auf den DOM angewendet, dies übernimmt die *Style Engine*
- Sind alle Style-Anweisungen ausgeführt worden spricht man von einen *computed style*
 - Wirken mehrere Anweisungen auf ein Element, so überschreibt die zuletzt angewendete Abweisungen die vorherigen



Architekturüberblick (Firefox Quantum)

Vereinfachter Ablauf: Erzeugen des Layouts

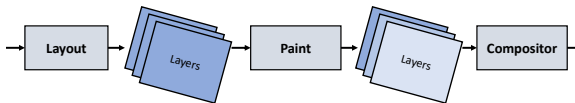
- DOM und ermittelte Style-Anweisungen werden an die Layout Engine übergeben
- Die Layout Engine berechnet das Layout basierend auf der gegebenen Fenstergröße
- Dabei wird jedes Element einzeln betrachtet, entsprechend dimensioniert und alle Style-Anweisungen angewendet



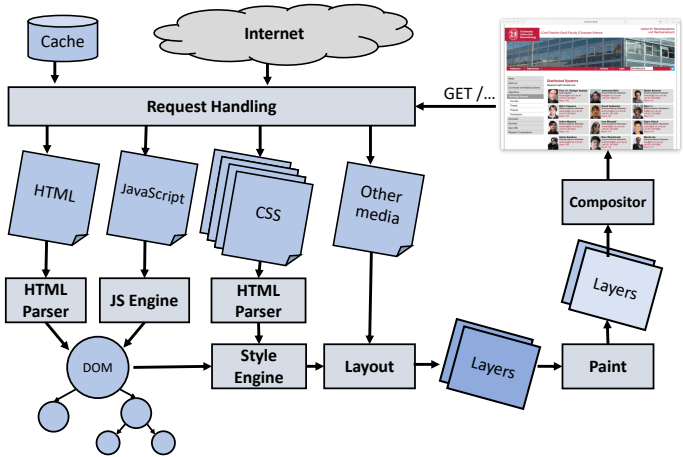
Architekturüberblick (Firefox Quantum)

Vereinfachter Ablauf: Darstellung im Browser

- Im finalen Schritt werden die berechneten Elemente mit allen Informationen dargestellt und die Seite wird für den Nutzer sichtbar
- Wenn der Nutzer den sichtbaren Bereich der Seite verändert muss neu gezeichnet werden
 - Tatsächlich passiert dies relativ häufig und Browser berechnen entsprechend mehr als den Viewport um schnell reagieren zu können



Architekturüberblick (Firefox Quantum)



Mehrprozessbrowserarchitektur & Isolation von Webanwendungen

Motivation

- Browser werden immer komplexer
- Webanwendungen nehmen ebenfalls in der Komplexität zu!
 - Beispiel: Single Page Applications (SPAs) und Progressive Web Applications (PWAs)
- Fehler in Browsern und Angriffe sind an der Tagesordnung
- Bedarf nach einer möglichst widerstandsfähigen Browser-Architektur

Ausgangspunkt

- Frühere Browser wurden als *Monolithen* konzipiert!
- Alle Architekturkomponenten haben vollen Zugriff
 - bspw. auf das Dateisystem
- Fehler können zum Absturz des gesamten Browsers führen
- Schlecht Isolation auch bezgl. Performance

Mehrprozessbrowserarchitektur & Isolation von Webanwendungen

Zielsetzung für die Mehrprozessarchitektur von Chromium

- Etablieren von Abstraktionen zur besseren Isolation der einzelnen Komponenten eines Browsers
- Isolationskonzepte müssen mit den existierenden Systemannahmen so kompatibel wie möglich sein
- Eine Mehrprozessarchitektur bietet bessere Sicherheit, Fehlertoleranz, Ressourcenverwaltung und Ausführungsgeschwindigkeit

Literatur

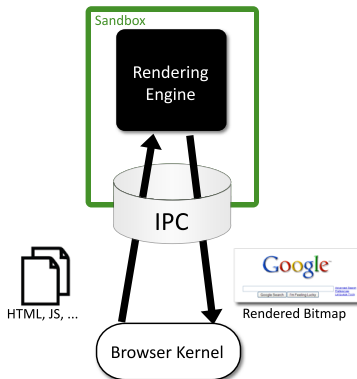
- *The Security Architecture of the Chromium Browser* [1]
- Weiterführende Details: <https://www.chromium.org/developers/design-documents/multi-process-architecture>
- *Isolating Web Programs in Modern Browser Architectures* [3]
- Weiterführende Details: <https://www.chromium.org/developers/design-documents/site-isolation>

Angreifermodell

- Entfernter Angreifer, der eine Lücke im Code des Browsers kennt
- Zur Ausnutzung der Lücke muss der Angreifer den Benutzer dazu bringen einen entsprechenden Schadcode zu laden
- Ziele des Angreifers
 - Installation von Schadsoftware wie bspw. auch Keylogger
 - Entwenden von wichtigen Daten/Dateien
- Angriffe die nicht betrachtet werden, weil sie nicht durch die Mehrprozessarchitektur adressiert sind
 - Phishing und Schwachstellen in Webanwendungen (z.B. für cross-site scripting (XSS))

Architekturüberblick

- Architektur wie um 2008 etabliert vgl. [1]



Funktionsaufteilung

- Rendering Engine
 - HTML/XML/XSLT/CSS parsing, Bilder dekodieren, SVG, Document Object Model (DOM), JavaScript Interpreter
 - Sonstiges: regulärer Ausdrücke

- Browser Engine
 - Verwaltung von persistenten Information
 - Zugriff auf native Grafikelemente bzw. den Fenstermanager
 - Oberfläche des Browsers (z.B. Location Bar)
 - Netzwerkfunktionen
 - *Achtung:* Die Browser Engine ist wesentlich weiter gefasst als in [2]

- Gemeinsame Funktionen
 - URL und Unicode parsing

Ziel der Aufteilung

- Rendering Engine enthält komplexen Code der Webanwendungen ausführt und eher fehleranfällig ist
- Browser Engine sammelt vor allem Code der direkt auf das System zugreifen darf

Sandbox

- Nimmt der Rendering Engine sämtliche Rechte um direkt auf das System zuzugreifen
- Stattdessen wird die Schnittstelle der Browser Engine für alle wichtigen externen Aufrufe genutzt

IPC (Inter-Process Communication)

- Schneller und effizienter Austausch zwischen Browser Engine und Rendering Engine
- Low-level IPC – ursprünglich über asynchrone *named pipes* realisiert
- (Mittlerweile eignes System Mojo welches auf verschiedene Subsystem abgebildet werden kann (z.B. Unix domain sockets))²

²<https://chromium.googlesource.com/chromium/src/+master/mojo/README.md>

Granularität der Aufteilung

- Vereinfacht kann (vorerst) angenommen werden das jedes *Fenster/Tab* einer eigenen Rendering Engine Instanz entspricht
- Auch gibt es wichtige Informationen, wie bspw. Fehlermeldungen zu HTTPS Zertifikaten oder Warnungen von Phishing-Versuchen, diese werden ebenfalls in einer eigenen Instanz erzeugt
- Ausnahmen gibt es bei Warnung zu dedizierten Seiteninhalten, diese werden in der bearbeitenden Rendering Engine erzeugt

Externe Hilfsprogramme (Plug-ins)

- Hilfsprogramme werden als eigener Prozess betrieben (evtl. auch in einer Sandbox)
- Es gibt in der Regel nur eine Instanz

Schnittstelle der Browser Engine

■ Rendering

- Alles wird gezeichnet und dann an die Browser Engine übergeben
- Browser Engine übergibt die Bilddaten an die entsprechenden Grafikschnittstellen zur Darstellung
- Im Rahmen der IPC bedeutet dies einen zusätzlichen Kopiervorgang (!)

■ Benutzereingaben

- Alle Benutzereingaben werden vom Betriebssystem an die Browser Engine übergeben
- Browser Engine wertet Eingaben entweder direkt aus oder gibt sie je nach Fokus an die Rendering Engine
- Rendering Engine erhält nur Ereignisse die sich direkt an die ausgeführte Webanwendung richten

Schnittstelle des Browser Engine (Fortsetzung)

- Persistenter Speicher
 - Aufgabe der Sandbox ist es zu erzwingen, dass die Rendering Engine (speziell im Fall einer Übernahme) nicht auf Dateien zugreifen kann
 - Entsprechende durch die Browser Engine initiierte Dialogfenster informieren den Nutzer über Up- und Downloads.
 - Im Falle von Downloads wird die Ablage von Dateien vorselektiert bspw. ein Standardablageverzeichnis
 - Zielsetzung ist zu verhindern das Dateien überschrieben werden oder ein Ablage an kritischen Stellen erfolgen kann

Schnittstelle des Browser Engine (Fortsetzung)

■ Netzwerk

- Jeglicher Netzwerkverkehr wird durch die Browser Engine abgewickelt
- Rendering Engine stellt entsprechende Anfragen an den Browser Engine
 - Handelt es sich um Protokolle wie `http(s)` und `ftp` lädt die Browser Engine die Daten und übergibt sie an die Rendering Engine
 - Bei `file` wird typischerweise blockiert, da die Rendering Engine sonst Zugriff auf das Dateisystem erlangt
 - Ausnahme wenn der Benutzer bspw. über die Location Bar explizit eine solche URL eingibt. In diesem Fall wird die URL über separate Rendering Engine Instanz abgewickelt

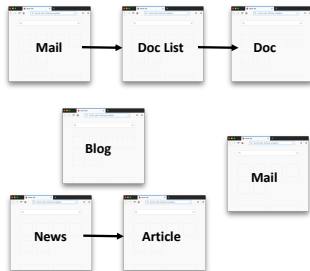
Evaluation

- Wirksamkeit wird überprüft mit Hilfe der Betrachtung von *Common Vulnerabilities and Exposures (CVE)*³
- Betrachtung von CVEs zwischen Juli 2007 und Juli 2008 für den Internet Explorer, Firefox und Safari
- Frage: Welche der veröffentlichten Schwachstellen können durch die vorgeschlagene Architektur vermieden werden?
 - Antwort: 38 von 87 der Rendering Engine zugeordneten Schwachstellen welche zur Ausführung von beliebigen Code geführt haben können verhindert werden
 - Dies entspricht 70.4% (38 von 54) von allen Schwachstellen welche zur Ausführung von beliebigen Code geeignet sind
- Mehr Details im technischen Bericht [1]

³<https://cve.mitre.org>

Isolation von Webanwendungen

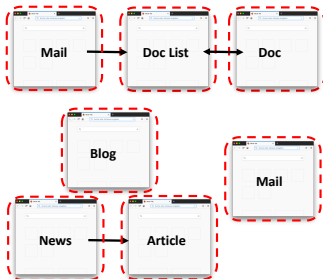
- Im Kontext eines Browsers werden verschiedene unabhängige Webanwendungen ausgeführt
- Wie kann man komplexe Webanwendungen auf eine Mehrprozessbrowserarchitektur abbilden?
- In der Vergangenheit alle Anwendungen in einem Prozess!



Isolation von Webanwendungen

Naiver Vorschlag

- Jedes Fenster/Tab erhält seinen eigenen Prozess
- Bricht Annahmen typischer Webanwendungen, welche Daten *miteinander* austauschen!
 - Gegenseitiger Zugriff auf den DOM nicht mehr möglich



Isolation von Webanwendungen

Zielsetzung

- Entwicklung einer Abstraktion welche Webprogramme identifizierbar macht und eine Isolation ermöglicht
- Gruppierung von zusammengehörigen Fenstern
 - Sollte den intuitiven Annahmen entsprechen
 - Kompatible zu existierenden Webanwendungen sein
- Idee: Ausnützen von existierenden Informationen
- Gruppierte Fenster sollen durch einen Prozess ausgeführt werden



Annäherung durch eine *ideale* Abstraktion

- **Webanwendung** (Web Program)
 - Menge von zusammengehörigen Seiten und zugeordneten Ressourcen die einen Dienst implementieren
- **Instanz einer Webanwendung** (Web Program Instance)
 - Eine laufende Instanz einer Webanwendung in einem Browser
 - Isolation erfolgt mit Unterstützung der Browserarchitektur

- Wie kann man das nun konkret umsetzen?

Isolation von Webanwendungen

Isolation auf Grund des Website

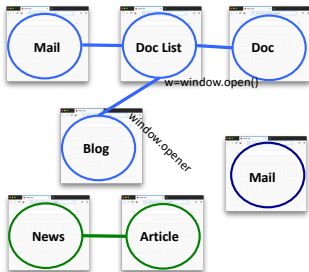
- *Same Origin Policy* erzwingt schon jetzt eine gewisse Isolation basierend auf Host, Protokoll und Portnummer
- Achtung: Webseiten können `document.domain` verändern (gilt nur für sehr alte Browser)
- Beschränkung nur im Rahmen des Domännennamen
- Website ist durch den Domännennamen, Protokoll und Port definiert



Isolation von Webanwendungen

Browsing Instanzen

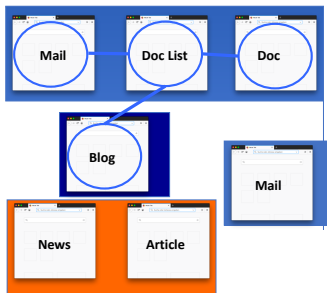
- Referenzen zwischen *verwandten* Fenstern
 - Eltern/Kind Relation durch `window.open()`
 - Zugriff auf den DOM der geöffneten Seite
 - Lebenszyklus eines Fensters
- Browsing Instanzen: Verbindung von Fenstern unabhängig vom Website



Isolation von Webanwendungen

Website Instanzen

- Schnittmenge von Website und Browsing Instanz
- Ermöglicht sichere Isolation von anderen Seiten
- Bildet eine kompatible Variante einer Webanwendung



Evaluierung

- Performance Isolation
 - Im Vergleich zu einer monolithischen Konfiguration reagiert der Browser unter Last viel schneller auf einen Klick
 - Verbesserte Geschwindigkeit durch Ausnutzung von Mehrkernsystemen
- Erzeugung eines Prozesses (100ms) ist bzgl. der Laufzeit nicht relevant
- Speicherbedarf nimmt spürbar zu, ist aber zu erwarten und ein akzeptabler Kompromiss

- Browser bestehen aus einer Vielzahl von komplexen Komponenten
 - Speziell die Rendering Engine ist kritisch, da sie entscheidet ob Webseiten einheitlich dargestellt werden
 - Aktuell gibt es wenige Kernsysteme die von verschiedenen Browsern genutzt werden (z.B. Webkit, Blink und Gecko)
- Ähnlich zu Betriebssystemen haben sich Mehrprozessarchitekturen auch bei Browsern durchgesetzt
 - Gründe sind Sicherheit, Fehlertoleranz aber auch Leistung
 - Weitere Informationen: Site Isolation: Process Separation for Web Sites within the Browser

<https://www.usenix.org/conference/usenixsecurity19/presentation/reis>

- Betrachtete Architektur Aspekte kratzen lediglich an der Oberfläche
 - Bspw. sind die internen Abläufe komplexer als dargestellt

Literatur

- [1] Adam Barth et al. *The Security Architecture of the Chromium Browser*. Jan. 2008.
- [2] Alan Grosskurth und Michael W. Godfrey. “A Reference Architecture for Web Browsers”. In: *Proceedings of the 21st IEEE International Conference on Software Maintenance*. ICSM '05. IEEE Computer Society, 2005, S. 661–664.
- [3] Charles Reis und Steven D. Gribble. “Isolating Web Programs in Modern Browser Architectures”. In: *Proceedings of the 4th ACM European Conference on Computer Systems*. EuroSys '09. Nuremberg, Germany: ACM, 2009, S. 219–232.