HTML5: Eine Sicherheitsanalyse

Sebastian Funke
funky324@yahoo.de
CASED TU Darmstadt

Überblick

Der Browser dient täglich für viele Menschen als selbstverständliches Werkzeug für alle Belange des Alltags. HTML5 gilt als Ansatz, den Webentwicklern die Arbeit zu erleichtern, Webseiten dynamischer zu gestalten und den Webnutzern noch mehr Interaktionsmöglichkeiten zu bieten. Doch worauf sollte man als Webentwickler und Browserbenutzer achten? In dieser Arbeit wird speziell auf den aktuell eingeführten HTML5-Standard eingegangen. Viele neue Features, wie zum Beispiel Cross-Domain-Kommunikation und Web Workers, wurden dem Standard hinzugefügt. Um die Benutzer und Web-Entwickler für die korrekte Verwendung der neuen Features zu sensibilisieren, werden bekannte Sicherheitslücken im Web genannt und die Auswirkungen von HTML5 auf diese beschrieben. Außerdem werden komplett neue Sicherheitsbedenken in Verbindung mit neuen HTML5 Features genannt und inwiefern diese schon in Browsern implementiert sind. Entwickler und Benutzer sollten aus aus den beschrieben Sicherheitslücken Konsequenzen im Umgang mit dem neuen Webstandard ziehen. Abschließend werden noch allgemeine Verbesserungsvorschläge für Browserhersteller vorgestellt.

1. EINFÜHRUNG

Das Internet wurde zu einem festen Bestandteil unserer Informationsgesellschaft. Kaum ein Mensch hat heutzutage keinen Zugriff auf das Internet und immer mehr Personen lernen die Vorteile des Internets zu schätzen, wie zum Beispiel vereinfachter Datenzugang von überall auf der Welt, Web-Applikationen mit denselben Möglichkeiten wie Desktop-Applikationen, Kommunikation über Foren, Chats, Social-Networks, u.v.m. Ob Unternehmern oder Privatpersonen, für jede Zielgruppe stehen vielfältige Möglichkeiten bereit ihre Interessen und Hobbys zu vertreten. Dabei hat sich ein Dienst des Internets besonders erfolgreich etabliert und zwar das umgangssprachliche „Surfen im Internet“. Dieser Service wird bereitgestellt durch Browser, die immer die aktuellsten HTML[1]-Features unterstützen müssen, welche von dem W3C[2] und den Browserherstellern vorgeschlagen, ausgearbeitet und übernommen werden. Dadurch können immer modernere Webseiten und Webservices von Webentwicklern zur Verfügung gestellt werden. Die vielen Implementierungsmöglichkeiten, die einem Webentwickler heutzutage zur Verfügung stehen, sind sehr komplex und vielfältig. Doch genau in dieser rasanten Entwicklung eines Mediums, welches immer mehr Benutzer umfasst, sehen Sicherheitsexperten viele sicherheitsspezifische Bedenken. Beispielsweise verschwimmt im Web immer mehr die klare Grenze zwischen Client und Server, die Möglichkeit von domain-übergreifender Webseiten-Kommunikation ermöglicht neue Angriffsvektoren und Benutzer selbst werden von den Experten als Problem angesehen [31]. Um die ausgehenden Gefahren von Drittherstellern-Browser-Plugins, wie zum Beispiel Flash, DivX usw. zu reduzieren und den Webentwicklern eine browserkompatible und sicherere HTML-Alternative zu bieten, wurden einige neue HTML5 Tags vorgesehen. Grafik- Video- und Multimedia Features sollen, statt von Plugins, von HTML5 übernommen werden. Darunter zum Beispiel der *<video>*-Tag, welcher das Einbetten von verschiedenen Videoformaten unterschiedlicher Kodierungen möglichst kompatibel erlauben soll oder der *<audio>*-Tag welcher analoge Funktionalität für Audio-Dateien bieten soll. Die **Same Origin Policy (SOP)**, welche zwar äußerst effektiv Browserbenutzer vor Zugriffsverletzungen, durch JavaScripte ihrer eigens aufgerufenen Webseiten schützt, aber in Projekten mit dynamischer Webseitenkommunikation sehr restriktiv Webentwickler eingrenzt, wird nun ergänzt um die **Cross Origin Policy** bzw. Cross-site HTTP Requests [27] [16]. Diese ermöglichen unter anderem, dass Webseiten miteinander im Browser kommunizieren können. Durch die Einführung von browser-unabhängigen **Web Workers**, den neuen Cache-Möglichkeiten, wie **Web Storage und WebSQL**, sowie vielen neuen APIs wie z.B. **Web Sockets, Media Capturing und Geolocation**, soll es nun möglich sein, die Grenze zwischen einfachen Webseiten und Desktop-Applikationen immer mehr verschwimmen zu lassen. Diese Entwicklung ermöglicht in Zukunft, dass immer mehr Applikationen und Ressourcen von überall, barrierefrei, gerät- und betriebssystem-unabhängig erreichbar sind. Diese Entwicklung bedeutet natürlich ebenfalls, dass HTML und dazugehörige APIs zu einer sehr sicherheitskritischen Schnittstelle werden. Aus diesem Grund beleuchtet diese Arbeit die oben genannten neuen Features etwas genauer, im Hinblick auf die Sicherheitsprobleme, die mit diesen einhergehen. Im nächsten Abschnitt werden einige allgemein bekannte Sicherheitsprobleme im Browser erläutert sowie die Auswirkungen dieser auf neue HTML5 Inhalte. Darauffolgend im Abschnitt 3 werden einige neue Features genauer analysiert. Abschließend wird im Abschnitt 3.11 der Implementierungsstand der davor genannten Features in aktuellen Browsern

[1]HTML: Hyper Text Markup Language
[2]World W3C: Wide Web Consortium

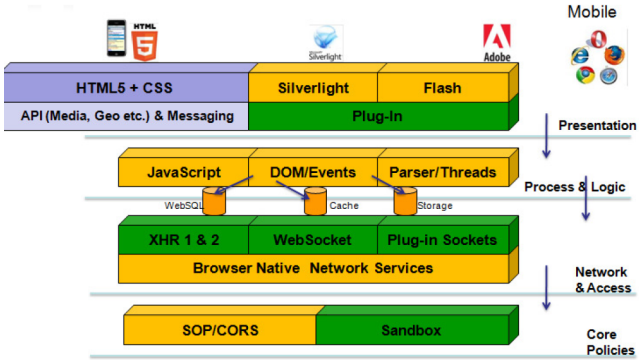

Abbildung 1: HTML5 im Browser[4]

aufgezeigt und es werden einige Verbesserungsvorschläge angemerkt, im Zusammenhang mit den zuvor erläuterten Problemen, aber auch allgemeine Vorschläge für die Browser-Hersteller zur Verbesserung der Sicherheit.

2. BROWSER-EVOLUTION: HTML5

Der HTML4-Standard unterlag seit 1997 einer langen Zeit der Entwicklungsstagnation. Die Arbeitsgruppen fixierten sich auf HTML-Erweiterungen wie XML, XHTML, CSS und dem kontinuierlichen Korrigieren und Verbessern von HTML4 Tags [28]. Erst 2004 hat sich eine Gruppe von Browserherstellern (Mozilla, Opera, Apple) zusammengeschlossen in der sogenannten WHATWG[3], um Vorschläge zu unterbreiten, welche dieser Entwicklung entgegenwirken sollten. Neben den browser-abwärtskompatiblen Verbesserungen an HTML4, welche vor allem dem Benutzer zu Gute kommen sollten, wurde besonderer Wert darauf gelegt, die statische HTML Natur dynamischer zu gestalten. Dies war vorher nur mit extensivem Einsatz von server- und client-seitigen Scriptsprachen, wie JavaScript und PHP, möglich. Abbildung 1 stellt die Implementierung diverser HTML5 Features in verschiedenen Browserschichten dar.

Schon in HTML4-Zeiten gab es sehr kritische Sicherheitsprobleme, mit denen Benutzer und Entwickler zu kämpfen hatten und immer noch haben. Die OWASP[5] listet dazu 2010, nach Aktualität und Verbreitung, die häufigsten Sicherheitsrisiken in ihren Top 10 Web Security Risks auf. Darunter die gefährlichen SQL[6] Injections [19], welche mehr in der unachtsamen Implementierung von server-seitiger Formulareingabe-Validierung liegen, als in HTML Fehlern, aber durch das neue HTML5 Feature, WebSQL, eine neue Angriffsfläche hinzubekommen. Dazu im Abschnitt 3.3 später mehr. Ein ähnliches Problem stellen RFIs und LFIs[7] dar, welche ebenfalls durch fehlende oder schlechte Eingabeprüfung ermöglicht werden. So können Angreifer beliebig injizierten Code per Browser auf dem Server der Webseite zur Ausführung bringen und damit die komplette Kontrolle über den Server erlangen. Diese sehr kritischen Sicherheitslücken werden, aufgrund der leichten Gegenmaßnahmen, immer seltener, jedoch finden die unterschätzten XSS-Angriffe (Cross-Site-Scripting) immer mehr Häufigkeit. Diese Schwachstellen basieren ebenfalls auf Code-Injektion. Der Unterschied ist der, dass XSS Angriffe gegen die Benutzer des Servers gerichtet sind und nicht gegen den Server. Der Code, der vom Angreifer auf dem Server eingeschleust wurde, wird dann im Browser des Benutzers ausgeführt, sobald dieser die Webseite mit dem injizierten Schadcode aufruft. Damit werden dem Angreifer zum Beispiel per HTML5 Cross Domain Kommunikation sensible Daten zugeschickt, wie z.B. die client-seitig gespeicherten Session IDs[8], welche ein Angreifer dazu missbrauchen kann, sich bei einer Webseite als das Opfer auszugeben. XSS Lücken können auch für sogenannte Phishing Angriffe genutzt werden. Eine Möglichkeit dazu ist Tabnapping[21], wobei ausgenutzt wird, dass das Opfer viele Browsertabs geöffnet hat. Beim Verlassen des Tabs (ohne diesen zu schließen) ändert das Angreiferscript im Tab das Favicon[9] in ein bekanntes anderes (z.B. GMail-Icon) und den Tabinhalt zu einer manipulierten angreifer-gesteuerten Fake-Webseite (z.B. Fake-Login von GMail). Irgendwann öffnet das Opfer wieder den Tab, wundert sich, warum es in z.B. GMail ausgeloggt ist, loggt sich wieder ein und offenbart dem Angreifer seine Logindaten. Speziell im Bereich XSS kommen mit der Einführung von HTML5 einige neue Angriffsvektoren hinzu, wie man im nächsten Abschnitt sehen kann. Leider verwenden die meisten Webentwickler für Benutzereingaben nur Blacklists, mit nicht erlaubten HTML4 Tags und Attributen, um XSS Probleme einzudämmen, aber neue Tags wie *<video>*, *<audio>* etc. befinden sich natürlich bisher noch nicht auf aktuellen Blacklists. Dasselbe gilt für neue Attribute, wie *onInput*, *onFocus*, *onError* und so weiter. Diese Attribute, welche bei vielen Tags vorhanden sind, können von Angreifern für XSS Attacken missbraucht werden. Beispiele dazu finden sich im Abschnitt 3.1. Diese Risiken bestehen natürlich auch nach der Einführung von HTML5, aber der neue Web-Standard fügt für bekannte Sicherheitsprobleme, wie oben gesehen, auch neue Angriffsvektoren hinzu [20]. Es werden aber auch neue Sicherheitsmaßnahmen eingeführt, wie z.B. Cross-Origin-Policies mit durchdachten Sicherheitsmodellen, sodass bei richtiger Implementierung dieser Features im Browser und deren Konfiguration durch Benutzer oder Webentwickler solchen Attacken vorgebeugt werden kann.

3. SICHERHEITSKRITISCHE HTML5 FEATURES

In diesem Abschnitt folgen nun Beschreibungen verschiedener neuer HTML5 Features, Angriffe, die mit Hilfe dieser realisiert werden könnten, sowie die Auswirkungen auf die Opfer.

3.1 Neue HTML5 Tags

Neben vielen neuen APIs, stellt HTML5 auch eine Reihe neuer, in der Einführung erwähnter, Tags und Attribute zur Verfügung. Speziell im Hinblick auf altbekannte und unterschätzte XSS-Schwachstellen erweitern diese das Angriffsspektrum. Die meisten Webentwickler haben bisher nur kritische HTML4-Tags und Attribute auf ihren Blacklisten der unzulässigen Benutzereingaben, sodass die neuen

[3]WHATWG: Web Hypertext Application Technology Working Group
[4] Shreeraj Shah, Blackhat EU 2012 [22]
[5]OWASP: Open Web Application Security Project
[6]SQL: Structured Query Language
[7]RFI/LFI: Remote/Local File Inclusion
[8]von dynamischen Webseiten server-seitig generierter String, welcher lokal gespeichert wird und als temporärer Authentifizierungscode benutzt wird
[9]kleines Icon links in der Adressbar des Browsers

Tags und Attribute viele Änderungen in deren Webanwendungen bedeuten. Es ist im Allgemeinen besser Whitelists zu verwenden, also nur Eingaben erlauben, welche erwünscht sind, anstatt unerwünschte zu verbieten, aber auch Whitelists müssen nach dem Webtech Artikel von Carsten Eilers[4] geändert werden. Ein Beispiel für neue Angriffsvektoren sind zum Beispiel die neuen Multimedia-Tags[10].

3.1.1 Multimedia-Tags

Die neuen Multimedia-Tags *<video>,<audio>,<canvas>*, welche kompatible Wiedergabe von Multimedia-Elementen zur Verfügung stellen sollen, können beispielsweise über das *onerror* Attribut missbraucht werden, wie im folgenden Code demonstriert.

Listing 1: Video und Audio-Tag Code Injizierung

```
<video onerror="javascript:alert('Attack')"><source>
<audio onerror="javascript:alert('Attack')"><source>
```

Man denke zum Beispiel an eine Web-Plattform die es erlaubt Beiträge zu kommentieren. Falls diese Kommentarfunktion die eingegebenen Kommentare nicht korrekt prüft und die obigen HTML5-Tags zulässt, entsteht ein möglicher XSS Angriff. Der Angreifer benutzt in seinem Kommentar einen der neuen HTML5 Tags, gibt eine imaginäre Elementquelle an und ein Script im Event-Attribut „onerror“. Dieses Schadscript wird dann aufgerufen, sobald ein Opfer den kommentierten Beitrag liest. Selbstverständlich könnte das Angreiferscript beispielsweise statt alert('Attack'), was nur ein Nachrichtenfenster öffnet, auch das Laden einer vom Angreifer gesteuerten, schädlichen, Webseite ermöglichen und vieles mehr. Des Weiteren wurden auch für bestehende Tags neue Event-Attribute hinzugefügt[11], wie in Listing 2 beispielsweise demonstriert.

Listing 2: Neues Event Attribute für bekannten Tag

```
<form id=bsp onforminput=alert('Attack')>...</form>
```

Weitere kritische Attribute wären zum Beispiel *onUnload*, *formAction*, *onFocus* (in Verbindung mit *autoFocus* führt der Aufruf der Webseite zur automatischen Ausführung des injizierten Scripts) , *onHasChanged*, *poster* und *srcdoc*. Mehr solcher Beispiele finden sich im HTML5 Security Cheat Sheet von Mario Heiderich[6].

3.1.2 SVG Grafiken

HTML5 erlaubt die einfache Einbindung von, seit 1999 existierenden, SVG-Grafiken[12]. Diese dynamischen Grafiken sind hervorragend geeignet für Animationen um Webauftritte noch schöner und effektiver zu designen[29]. Die Einbindung erfolgt über den *<svg>* Tag. Folgender Aufruf zeichnet beispielsweise einen roten Kreis.

Listing 3: SVG Definition eines Kreises

```
<svg>
   <circle cx="100" cy="50" r="40" stroke="black"
   stroke-width="2" fill="red" />
</svg>
```

Neben der Möglichkeit in einer SVG Datei eine Grafik zu beschreiben, kann man auch allerlei Inhalte wie Verlinkungen und JavaScript einbauen, wie Mario Heiderich 2011 in einer Präsentation gezeigt hat [7]. Da das Verwenden von SVG Dateien vielseitig möglich ist, z.B. auch im *<img>* Tag, als CSS Hintergrundbild, im *<iframe>, <object>, <embed>* Tag oder schlicht ausführbar als Bilddatei auf dem Desktop, ermöglichen SVGs sehr vielschichtige Angriffsmöglichkeiten. Diese werden auch aktiv ausgenutzt zum Beispiel von Pornoseiten oder in Emailanhängen. Ein von Mario Heiderich beschriebener lokaler Angriff wäre, beispielsweise über eine präparierte SVG Datei mit einem für das Opfer unterhaltsamen Inhalt möglich. Wird die Datei von dem Opfer heruntergeladen und lokal, mit einem Doppelklick, erneut geöffnet, könnte sich ein eingebautes JavaScript aktivieren, die Dateien in dem Verzeichnis des gespeicherten SVGs stehlen, Unterordner dieses Verzeichnisses auslesen oder sogar Java Applets starten. Somit bieten SVG Dateien viele Angriffsvektoren, die von vielen Entwicklern unterschätzt werden.

3.1.3 Drag und Drop API

Events, wie *onDrag* bzw. *onDrop* kann ein geschickter Angreifer mit etwas Social-Engineering wirkungsvoll für XSS Angriffe ausnutzen, wie Krzysztof Kotowicz anschaulich in einer Demo 2011 gezeigt hat[10]. Allgemein wurde die Drag und Drop API dahingehend erweitert, dass es auch möglich ist Dateien von außerhalb des Desktops in den Browser zu ziehen, was viele neue fragwürdige Benutzerinteraktionen ermöglicht. Zwei denkbare Angriffsszenarien sind in der Arbeit „HTML5 Overview: A Look at HTML5 Attack Scenarios“, der Sicherheitsfirma TrendMicro beschrieben [15]. So ist es beispielsweise durch SE[13] möglich, dass Drag'n'Drop Daten in der Zwischenablage vom Angreifer manipuliert werden können über *onDragStart= „event.dataTransfer.setData('Changed Text')“*, was beim Drag und Drop eines unsichtbaren DIV-Containers geschehen könnte. In einem anderen Szenario könnte der Angreifer zum Beispiel in einem versteckten Iframe eine Webseite mit sensiblen Daten über das Opfer laden (sofern dieses Opfer mit Session ID den temporären Zugang dazu gelegt hat) und das Opfer auffordern diesen Inhalt, auf eine vom Angreifer kontrollierte Webseite, per Drag und Drop zu verschieben. Wie oben erwähnt, könnte über SE auch ein Angreifer den Benutzer dazu bringen, geheime Daten vom Desktop auf die Angreiferseite zu schieben, sodass diese dem Angreifer zugesendet werden. Daher sollten Benutzer sich der Risiken von Drag und Drop bewusst werden und sensibilisiert werden für se-basierte Angriffe.

3.1.4 Unabhängige Formularelemente

In der Vergangenheit gehörten alle Formularelemente in ein umschließenden *<form>...</form>* Tag. Mit HTML5 sind auch Formular-Elemente außerhalb davon möglich, indem die Elemente einfach über das Attribut *form=„formID“* einem Formular mit der ID „formID“ zugeordnet werden können. Somit ist es möglich, dass per SE aufgefordert wird,

[10] `http://www.w3schools.com/html5/html5_new_elements.asp`

[11] `http://www.w3schools.com/html5/html5_ref_eventattributes.asp`

[12] SVG: Scaleable Vector Graphic, XML beschriebenes, zweidimensionales Bildformat

[13] SE: Social Engineering

ein Button oder Bild zu klicken, welcher bzw. welches das zugehörige Formular manipuliert oder sensible Daten aus dem Formular kopiert, um diese einem Angreifer zu schicken. Der Angreifer kann sogar ohne Benutzerinteraktion über *onFormInput*- oder *onFormChange*-Events Benutzerdaten ausspähen und vieles mehr, was beispielsweise durch neue Button-Attribute wie *formAction*, *formMethod*, *formNoValidate* und *formTarget* möglich ist[14]. Ein weiteres neues Attribut des <*form*> Tags ist **autocomplete**[15]. Mit eingeschaltetem Attribut reagiert das Formulareingabefeld auf Eingaben mit Eingabevorschlägen für den Benutzer. Da diese vorgeschlagenen Daten oft sensible Daten wie Adresse oder gar Passwort sind, scheint auch hier eine Lücke zu bestehen. Diese Daten können aber nicht vom Angreifer aus dem DOM[16] geladen werden, weil diese in temporären Drop-Boxen erscheinen. Dennoch kann sich auch hier der Angreifer mit etwas SE behelfen, wie Lavakumar Kuppan in einer Demo gezeigt hat [11]. Darin schlägt er dem Opfer ein Spiel vor, bei dem das Opfer kontinuierlich die Enter Taste drücken muss, während das Angriffsscript im Hintergrund eine versteckte Textbox mit aktiviertem *autocomplete*-Attribut mit Buchstaben füllt. Daraufhin treten verschiedene Vorschläge auf, die per Enter-Taste übernommen werden und dann ausgelesen werden können. Wie man sieht steckt der Teufel oft im Detail.

3.1.5 Sandboxed Iframes

Eine weitere weitverbreitete Angriffsmöglichkeit ist Clickjacking oder UI Redressing. Dabei werden die Inhalte von Webseiten dahingehend manipuliert, dass Benutzer unfreiwillig bestimmte Klicks machen. Beispielsweise ist es möglich mit einem JavaScript ein <*iframe*> kontinuierlich unter dem Mauszeiger zu positionieren. Falls das Iframe nun über das *opacity* CSS-Style-Attribut noch unsichtbar gestellt ist und ein *OnClick* Ereignis ausgelöst wird, beispielsweise in der Webseite des Iframes, kann der Angreifer damit den Benutzer zwingen, bestimmte ungewollte Klicks auf beliebigen, eingebetteten Webseiten zu tätigen[3]. Viele Webseiten setzen ähnliche Tricks auch gezielt für Werbebanner-Klicks ein. Clickjacking gab es allerdings schon vor HTML5. Welche Rolle spielt dabei nun HTML5? Der beschriebene Iframe-Tag hat mit HTML5 ein Sicherheits-Attribut-Upgrade bekommen und zwar das ***sandbox*** Attribut[17]. Dieses kann die Rechte des Iframe-Inhalts beschränken oder ausdehnen und ermöglicht als Eingaben: einen leeren String („“ bedeutet volle Restriktion, also keine Formulare in Iframes, keine Kommunikation mit anderen IFrames oder der Elternseite und keine Scriptausführung im Iframe), allow-forms, allow-same-origin (erlaubt die Kommunikation des Iframe-Inhalts mit der Elternseite), allow-scripts, allow-top-navigation oder mehrere der genannten Werte mit Leerzeichen getrennt. Damit hat man zum einen bei falscher Handhabung eine große Lücke, indem man domain-übergreifende Kommunikation über das Iframe erlauben kann, zum anderen kann man allerdings das Iframe auch komplett absichern. Dadurch können aber keinerlei Daten das Iframe verlassen und Scripte im Iframe nicht ausgeführt werden. Eine Möglichkeit Click-Jacking zu vermeiden, welche unter anderem von Facebook eingesetzt wird, ist sogenanntes Framebusting oder Frame-killing [5] [18]. Allerdings wird dieses Verfahren ausgehebelt, wenn in dem Sandboxed-Iframe *Allow-Script* nicht aktiviert ist. Viele weitere anschauliche Beispiele, auch in Verbindung mit XSS Lücken, finden sich in der Präsentation „HTML5: Something Wicked This Way Comes“ von Krzysztof Kotowicz [10].

3.2 Web Messaging und CORS über XHR

Die strikte Implementierung der SOP erlaubt keinerlei dynamische JavaScript-Kommunikation einer Webseite mit anderen Quellen, auch nicht mit Frames. Bisher konnte diese Kommunikationsbeschränkung nur von Drittherstellerplugins, wie Flash, Silverlight, JSONp oder server-seitigen Proxys, aufgehoben werden. Mit dem Web-Messaging bzw. der Cross-Document-Messaging API (Kommunikation zwischen JavaScript **window**-Objekten, domain-übergreifend mittels des *postMessage()*-Befehls)[19] und CORS[20] wurden in HTML5 zwei neue Möglichkeiten eingeführt, diese Beschränkung, ohne Plugins anderer Hersteller, zu umgehen. Beim Web-Messaging ist client-seitige Window-Objekt- und domain-übergreifende Kommunikation dynamisch, mit Hilfe des Javascript-Befehls *postMessage(...)*, möglich[21]. Bevor eine solche Kommunikation entstehen kann, muss das sendende Element das Window-Objekt des Empfängers kennen. Außerdem steht dieses Feature nur zur Verfügung, wenn:

- das Ziel ein Frame im sendenden Window ist
- das Ziel ein vom Senderwindow geöffnetes Window ist
- das Ziel das Elternelement des Senders ist
- oder es das TopLevel des Elternelements ist

Im Falle, dass ein zuhörendes Window-Element auf eine Nachricht wartet und beim Eintreffen einer Nachricht die Herkunft nicht oder falsch prüft oder die Herkunft einer Nachricht vom Angreifer gefälscht wurde, kann diese manipulierte Angreifernachricht die Arbeitsweise der Webanwendung beeinflußen. Weitere Angriffe mittels Web Messaging wurden 2012 auf der BlackHat EU von Shreeraj Shah beschrieben [22] und in der Arbeit „The Emperors New Apis“[24]. Für die Verwendung von postMessage findet sich ein Beispiel im Blog von Robert Nyman[18].
Ein weiterer neuer Weg zur Cross-Domain-Kommunikation, um Flexibilität für Webentwickler zu schaffen, ist das in HTML5 eingeführte CORS. Cross-Domain Requests Sharing über XMLHttpRequests[22]. Bei Cross Domain Requests wird unterschieden zwischen „simplen“ und „nicht simplen“ CORS Requests [31]. „Simple“ Requests werden ausgeführt ohne Rücksicht auf die Bereitschaft der anderen Seite, ohne Handshake, was Angreifer über automatische Formular-Submits ausnutzen können. Der essentielle Sicherheitsmechanismus sieht vor, dass bei einer Antwort auf eine Anfrage im Header das Attribut

[14] `http://www.w3schools.com/html5/tag_button.asp`
[15] `http://www.w3schools.com/html5/att_form_autocomplete.asp`
[16] DOM: Document Object Model
[17] `http://www.w3schools.com/html5/tag_iframe.asp`
[18] ein Script prüft, ob die Seite in einem Frame aufgerufen wurde und läd sich in diesem Fall selbst neu in dem Elternelement des Frames
[19] `https://developer.mozilla.org/en/DOM/window.postMessage`
[20] CORS: Cross Origin Resource Sharing
[21] `http://www.w3.org/TR/webmessaging/`
[22] API zum Datentransfer über HTTP/HTTPS Protokoll

Access-Control-Allow-Origin: xxx angegeben ist. Würde ein Webentwickler dort das Wildcard * angeben, statt des Domainnamens für xxx, wäre Kommunikation mit jeder Domain erlaubt, sodass für einen Angreifer alle Türen geöffnet wären[8]. Im folgenden Codeausschnitt wird eine solche Kommunikation illustriert.

Listing 4: XMLHttpRequest von http://site.com

```
var x = XMLHttpRequest();
x.open('GET','http://site.com/get_message.php?id=23'
,false);
x.send(null);
```

Daraus entsteht ein HTTP Request, welcher grob in Listing 5 dargestellt ist.

Listing 5: XMLHttpRequest Ergebnis

```
GET /get_message.php?id=23 HTTP/ 1.0
Host: site.com
Cookie: SITE_SESSION_ID=0123456789ABCDEF
Origin: http://www.site.com
```

Damit die Antwort des Requests über Domaingrenze hinweg gelesen werden kann muss der Server antworten mit:

Listing 6: Serverantwort

```
HTTP/1.0 200 OK
Access-Control-Allow-Origin: http://www.site.com

Geheime Nachricht: "Hallo andere Webseite"
```

Ein Angreifer könnte eine CSRF[23] Attacke starten. Also das Unterschieben eines manipulierten HTTP Requests über eine manipulierte URL oder über XSS.
In einem Beispielszenario könnte so einem Opfer, welches angemeldeter Administrator einer Webseite ist, ein HTTP Request untergeschoben werden (z.B. indem das Opfer auf eine mit XSS manipulierten Seite surft, sodass der HTTP Request per Angreiferscript ausgeführt wird). Der untergeschobene HTTP Request könnte nun mit den Rechten des eingeloggten Administrators einen weiteren Administratoraccount auf der Webseite erstellen und somit die Webseite des Administrators bzw. Opfers kompromittieren. Auf der BlackHat EU 2012 wurde von Shreeraj Shah zu diesem Thema die Silent CSRF Attacke beschrieben. Die Abbildung 2 verdeutlicht das Vorgehen des Angreifers (für Details siehe [22]). In späteren Versionen von CORS wurden „nicht simple“ Requests eingeführt, mit Preflight[25], welche die oben genannten Attacken abschwächen[26]. **Reverse Web Shell** oder Shell of the Future, was 2010 von Lavakumar Kuppan auf der BlackHat Konferenz vorgestellt wurde, ist ein weiterer sehr anschaulicher Ansatz, inwieweit XSS Sicherheitslücken für Session Sidejacking, in Verbindung mit CORS, ausgenutzt werden können. Dazu werden über getunnelte COR-Kommunikation, Session IDs, welche über XSS ausgespäht wurden, ausgenutzt, um mit den aktuellen Browserrechten von Opfern auf den angegriffenen Webseiten zu agieren[13]. Im Worst-Case kann ein Angreifer über diese getunnelte Session, das Laden einer weiteren manipulierten Webseite einleiten, welche einen Browserexploit ausnutzt. Auf diesen Exploit wartet zum Beispiel ein Metasploit-Server[26], sodass der Angreifer über diesen Exploit eine firewall-resistente Remote-Reverse-TCP-Shell Verbindung aufbauen kann. Die Session kann er dann beispielsweise in die *explorer.exe* (sofern das Opfer Windows benutzt) integrieren, sodass die Verbindung auch unabhängig vom geöffneten Browser bestehen bleibt, der Angreifer volle Kontrolle über das Opfersystem hat und sogar die Kontrolle auf weitere Systeme im lokalen Netzwerk des Opfers ausbauen kann. Weiterhin ermöglicht CORS das Senden von willkürlichem Inhalt über URL Anfragen, da viele Webanwendungen das Verarbeiten folgender URLs erlauben:

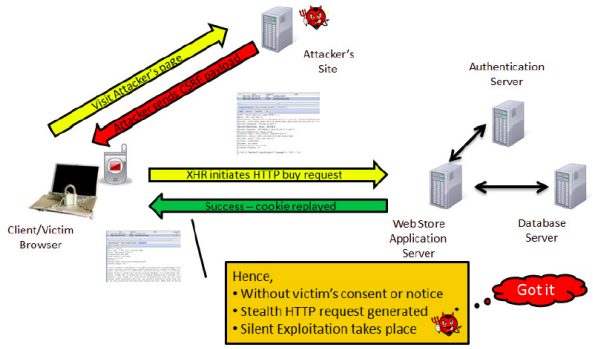

Abbildung 2: CSRF mit XHR/HTML5[24]

Listing 7: URL XSS durch COR Policy

```
http://www.example.com/#index.php
http://www.example.com/index.php?page=example.php
```

Damit sind ebenfalls neue Ansätze für XSS Attacken geschaffen, solange Webentwickler übermittelte GET Werte nicht genauer überprüfen [15].
Anwendung findet CORS allgemein überall wo AJAX[27] Technologie in Verbindung mit dynamischer Kommunikation verwendet wird, also auf Webservices wie *Facebook, Google Maps, Gmail.*

3.3 Web Storage

Viele Webentwickler, zum Beispiel im Bereich Browsergames, hatten vor HTML5 den Wunsch mehr Daten persistent, lokal auf den Rechnern der Benutzer speichern zu können. Dieser Wunsch wurde mit dem neuen Feature, Local Storage und Web Storage erfüllt, welches die Restriktionen, von herkömmlichen Methoden wie Cookies, aufhebt. Die Local Storage API speichert alle Daten in einem einfachen Schlüssel-Werte-Paar, welches leicht über JavaScript gespeichert und abgerufen werden kann. Des Weiteren können auch ChangeListener für den Local Storage registriert werden[28]. Im folgenden Codeausschnitt (Listing 8) wird dieses Feature demonstriert.

[23] CSRF: Cross-Site Request Forgery
[24] Shreeraj Shah, Black Hat 2012 [22]
[25] zweiwegiges Handshake (Origin,Access-Control-Request-Method,Access-Control-Allow-Headers) um Bereitschaft des Servers vorher zu prüfen
[26] `http://www.metasploit.com/`
[27] AJAX: Asynchronous JavaScript and XML
[28] `http://www.w3.org/TR/webstorage/`

Listing 8: Local Storage

```
<script>
 localStorage.setItem("ItemName","ItemValue");
 var bsp = localStorage.getItem("ItemName");
</script>
```

Ein weiterer Ansatz ist WebSQL, um Daten persistent und lokal in einer Datenbank zu speichern, was bessere Offline-Kompatibilität zu Webapplikationen sicherstellen soll. Allerdings ist dieses Feature seit 2010 kein Teil der HTML5 Spezifikation mehr[29]. Es wurde allerdings trotzdem von Chrome, Opera und Safari implementiert. Weitere Informationen zu WebSQL findet man auf der Webseite HTML5Doctor.com[23]. Auch dieses Feature bleibt nicht von Angriffen verschont. Beispielsweise kann über XSS oder DNS Spoofing, ein Angreifer, auf die lokal gespeicherten Daten, zugreifen[30]. Ebenfalls denkbar, wie in der Einführung schon beschrieben, sind altbekannte SQL-Injection Attacken auf die WebSQL Datenbank. Diese ermöglichen einem Angreifer, durch geschickt gewählte Eingaben, die gesamte lokale Datenbank auszulesen, sofern die Eingaben oder allgemein, die Zugriffsschnittstellen, nicht ordnungsgemäß geprüft werden. Da viele Webapplikationen mit Datenbankanbindungen sensible Daten speichern, welche oftmals nicht einmal kryptografisch verschlüsselt sind, wirken sich solche Lücken umso kritischer aus. Anschauliche Demos dazu gibt es auf andlabs.org von Lavakumar Kuppan[12]. Außerdem ist zwar der Local Storage und WebSQL Speicher auf 5 MB beschränkt, jedoch erlaubt der Safari Browser beim Überschreiten dieses Limits, das Aufheben dieser Beschränkung. Damit kann ein Angreifer den Speicher eines unwissenden Opfers, welches die Beschränkung ohne nachzudenken aufhebt, mit unnötigen Daten verbrauchen, was speziell im Hinblick auf mobile Endgeräte mit Browsern kritisch wäre.

3.4 Offline Web Application Cache

Viele Webanwendungen benötigen heutzutage erweiterten lokalen Speicher, wie im oberen Abschnitt 3.3 beschrieben und da solche Anwendungen immer komplexer werden und zuverlässiger sein müssen, sollte es möglich sein, diese auch offline zu speichern. Dadurch sind diese ohne Internet ausführbar und können alle geänderten Daten auf dem Server aktualisieren, sobald wieder eine Internetverbindung besteht. Diesen Ansatz verfolgen die neuen Offline-Caching Möglichkeiten von HTML5. Die Spezifikation und Beispiele findet man auf der Webseite der WHATWG[30] oder der W3-School[31]. Jeder Webseite ist es möglich eine Liste von URLs in der sogenannten *manifest* Datei zu speichern. Der Browser speichert alle Dateien, welche darin mit URL angegeben sind, lokal, hält diese immer aktuell und sobald der Zugriff auf die Webseite nicht möglich ist, verwendet er die lokale Version. Zum Cachen einer Seite muss lediglich folgendes Attribut im *<html>* Tag gesetzt werden.

Listing 9: Caching mit Manifest Datei

```
<html manifest="/CacheDatei.manifest">
```

[29] http://dev.w3.org/html5/webdatabase/

[30] http://www.whatwg.org/specs/web-apps/current-work/multipage/offline.html

[31] http://www.w3schools.com/html5/html5_app_cache.asp

Es muss noch beachtet werden, dass der Content Type der Manifestdatei auf *text/cache-manifest* gesetzt ist. Damit hat diese Datei folgende Form:

Listing 10: Caching mit Manifest Datei

```
CACHE MANIFEST
/index.html
/images/bild.png
```

Eine Angriffsmöglichkeit auf dieses Konzept wäre sogenanntes Cache-Poisoning, also das Cachen von falschen, manipulierten oder schädlichen Webseiten oder Ersetzung von gecachten Webseiten durch andere. Das führt bei einer WebMail-Anwendung zu gefährlichen Phishing-Angriffen, wie in einem andLab.org Blogeintrag beschrieben ist[1].
In einem denkbaren Szenario eines unsicheren, offenen W-LANs, könnte der Angreifer sich als, vom Opfer aufgerufene, Webseite ausgeben. In dieser Situation kann er dann Anfragen mit einer eigenen Version der Webseite beantworten, welche daraufhin im Cache des Opfers gespeichert wird. Somit bleibt die Fake-Webseite im Cache erhalten, selbst nachdem das Opfer das WLAN verlässt. Diese Attacke ist auch mit dem gewöhnlichen Browsercache möglich, jedoch erlaubt der HTML5 Application Cache sogar Root-Caching, wodurch im Vergleich zum gewöhnlichen Cache auch HTTPS Verbindungen erfolgreich vergiftet werden können. Die genauen Details, eine POC[32]-Attacke und Möglichkeiten das Cachen persistenter zu halten, werden ebenfalls von andlab.org beschrieben.

3.5 Web Workers

Um die Performance zu steigern, führt HTML5 sogenannte Web Workers[33] ein. Dabei handelt es sich um Javascript, welches in eigenständigen Threads, unabhängig von den Scripten zur Anzeige der Oberfläche der Webseite, nebenläufig ausgeführt wird. Wodurch Verzögerungen durch komplexe Berechnungen sozusagen in den Hintergrund ausgelagert werden können, sodass nur noch die Verzögerung der eigentlichen Webseite bleibt. Beispielsweise könnten aufwendige Transformationen, mehrdimensionaler Grafiken, in mehreren Threads parallel berechnet werden und anschließend in der Canvas Umgebung ausgegeben werden. Web Worker kommunizieren über Nachrichtenweiterleitung, da sie keinen Zugriff auf das DOM ihrer aufrufenden Webseite haben. Die Deklaration eines solchen Web Workers ist einfach per JavaScript möglich, wie im folgenden Ausschnitt demonstriert.

Listing 11: Web Worker deklarieren und starten

```
var worker = new Worker('task.js');
worker.postMessage(); // Start the worker
```

Mithilfe von DOM basierten XSS Angriffen kann dieses sehr nützliche HTML5 Feature allerdings auch als KeyLogger genutzt werden, wie in der Präsentation der BlackHat EU 2012 gezeigt wurde[22]. Außerdem unterstützen Web Workers die Idee eines browser-basierten Botnetzes, welches den Vorteil mit sich bringt, betriebssystem- und plattform-unabhängig zu sein. Des Weiteren bietet es eine große Angriffsfläche,

[32] POC: Proof Of Concept

[33] http://www.w3.org/TR/workers/

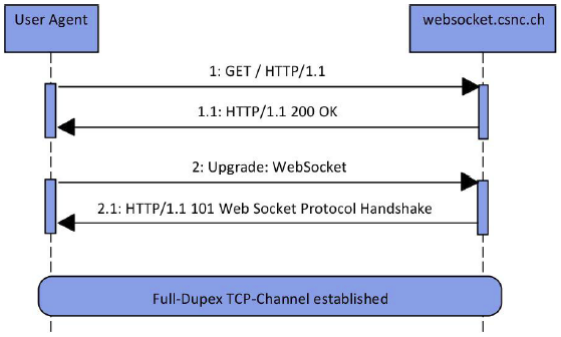

Abbildung 3: Web Socket Handshake[37]

da es eine Vielzahl ungeschützter und javaScript-exploitanfälliger Systeme gibt. Weiterhin laufen Botscripte damit unauffällig und unsichtbar im Hintergrund des Browsers. Die schlechte Persistenz des Botscripts kann nach Kuppan über die Infizierung mithilfe von persistenten XSS Lücken und Clickjacking in Verbindung mit Tabnapping verbessert werden[13]. Die Einsatzmöglichkeiten eines solchen dynamischen Botnetzes sind sehr vielseitig. Diese reichen von DDos Angriffen (beispielsweise mit COR-GET-Requests), über Spamming (Emailspamming über schlecht konfigurierte Formulare), Bitcoin[34] Generierung, Phishing, Scannen interner Netzwerke, bis hin dazu sich wurmartig selbst weiterzuverbreiten (über XSS oder SQL Angriffe). Damit hat ein Angreifer zum Beispiel über Netzwerk-Proxy Nutzung, im Sinne des Reverse Web Shell, ein sich ständig vergrößerndes, schwierig zurückzuverfolgendes Netzwerk an flexiblen Botbrowsern, zur Verfügung.

3.6 Web Socket API

Die Web Socket API[35], ursprünglich von der IETF[36] entworfen, bietet eine getunnelte bidirektionale, vollduplex Kommunikation über ein TCP Socket und wird schon von den meisten Browsern unterstützt. Es existieren sogar Multiplayer-Browsergames, beispielsweise von Mozilla, welche Gebrauch von diesem Feature machen[38]. Neben vielen produktiven Einsatzmöglichkeiten, kann die API auch zum Port-Scanning verwendet werden[15]. Dazu prüft man den *Connection-ReadyZustand* eines Verbindungsaufbaus und die Zeiten zwischen dem Übergang einer solchen Zustandsänderung.
XMLHttpRequests haben ähnliche Zustände, welche man ebenfalls abfragen kann und für solch einen Zweck verwenden könnte. Die Genauigkeit dieses Verfahrens wird allerdings nie an einen üblichen Portscanner, wie NMap[39], herankommen, da ein Web Socket Port Scanner auf der obersten Schicht des OSI-Schichtenmodells angesiedelt ist. Dennoch erlauben solche Portscanner das Scannen von Rechnern im Netzwerk des Opfers, da die Verbindung getunnelt an der Firewall vorbeigeführt werden kann, was mit üblichen Portscannern nicht ohne weiteres möglich wäre. Das Tool „JS-Recon“ von Kuppan demonstriert einen solchen Scanner[40]. Neben dem Portscannen ist es auch möglich, Opfer nach Browserschwachstellen zu scannen. Dazu wird die 2006 von Wade Alcorn skizzierte Interprotocol Communication [25] verwendet, was man in der Praxis bei dem Browser-Vulnerability-Scanner BEEF[41] sehen kann. Auch eine Remote Web Shell ist über Web Sockets möglich, welche auch nach dem Schließen des Browsers bestehen bleibt. Auf diese Weise ist es für den Angreifer nicht nur möglich, im Domain-Kontext des Opfers zu surfen, sondern aufgrund der Persistenz sogar ganze BotNetze aufzubauen. Eine POC-Demo zu diesem Problem wird in der HTML5-Web-Security Arbeit der Compass Security AG präsentiert [17].
In der Arbeit „Talking To Yourself For Fun And Profit“, der Carnegie Mellon University wird weiterhin beschrieben, wie es unter anderem mit Web Sockets möglich ist,
Cache Poisoning Angriffe gegen Proxyserver auszuführen. Somit können im Worst Case ganze Netzwerke, die den Proxyserver verwenden, mit Schadcode infiziert werden[14].

3.7 Desktop Notification API

Ein weiteres sehr interessantes neues HTML5 Feature ist die Desktop Notification API. Diese lässt außerhalb des Browsers individuelle Benachrichtigungen zu, welche allerdings im groben Aufbau von der Implementierung im Browser abhängen und vom Benutzer erlaubt werden müssen[42]. Dieses Feature ermöglicht Phishing Attacken, da die inhaltliche Nachrichtengestaltung sogar per HTML möglich ist. Zum Beispiel könnte ein Angreifer eine Re-Login-Notification-Nachricht von *Gmail* fälschen, auf die ein Opfer hereinfallen könnte. Auch wenn in der Nachrichtenbox oben links immer noch die Herkunftsdomain angezeigt wird, ist es denkbar, dass dieses Feature für Phishing missbraucht wird. Ein ähnliches Problem stellen falsche Anti-Virus-Meldungen da, die von Angreifern über dieses Feature generiert werden könnten.

3.8 History API

Die Browserhistory speichert alle besuchten URLs und ermöglicht über die Befehle *back()*, *forward()* und *go()* die Vorwärts- und Rückwärtsnavigation im Browser. HTML5 führt zwei neue History-Befehle ein ***pushState(data, title, [url])*** und ***replaceState(data, title, [url])*** [43]. Es ist also möglich, per JavaScript manuell History-Elemente hinzuzufügen. Diese Funktionalität kann von einem Angreifer dazu missbraucht werden, einem Opfer fragwürdige URLs unterzuschieben, um diese später damit zu erpressen. Ebenfalls besteht die Möglichkeit, dass der Zurück-Button des Browsers über *pushState()* Aufrufe manipuliert werden kann, sodass bei einem Klick auf Back, die Seite nicht verlassen werden kann. Mit der *replaceState()*-Funktion ist es möglich beliebige History-Einträge durch URLs mit der aufrufenden Domain zu ersetzen. Mit etwas Geschick ist es somit möglich, Phishing Attacken auszuführen, indem die gespeicherte URL der Bank geändert wird, wie im folgenden Beispiel gezeigt.

[34] Cyberwährung: http://bitcoin.org/
[35] `http://www.w3.org/TR/websockets/`
[36] IETF: Internet Engineering Task Force
[37] Shreeraj Shah, Blackhat EU 2012 [22]
[38] `http://hacks.mozilla.org/2012/03/browserquest/`
[39] `http://nmap.org/`
[40] `http://www.andlabs.org/tools/jsrecon.html`
[41] `http://beefproject.com/`
[42] `http://www.w3.org/TR/notifications/`
[43] `https://developer.mozilla.org/en/DOM/Manipulating_the_browser_history`

Listing 12: Phishing Angriff über Historyänderung

```
<script>
 history.replaceState({},"",
 "/www.sparkasse.de/login.php");
</script>
```

Falls also ein Sparkassen-Online-Banking Benutzer das obige Script ausführt, wird seine Sparkassen-History geändert in:
www.angreifer.de/www.sparkasse.de/login.php, sodass das Opfer beim Aufruf des Sparkassen-Logins, aus der History, auf eine vom Angreifer manipulierte Phishing-Webseite gelangt, wo es seine Logindaten unwissentlich an den Angreifer weitergibt[44].

3.9 Geo Location API

Eine weitere Neuerung ist die vielseitig einsetzbare Geo Location API, welche es dem Browser erlaubt, relativ genau, die geographische Position des Benutzers zu bestimmen. Natürlich ist solch ein Feature ein großer Einschnitt in die Privatsphäre eines Nutzers, sodass dieses Feature auf einer Webseite nur mit ausdrücklicher Erlaubnis des Benutzers genutzt werden darf. Auf HTML5Demos[45] findet sich dazu eine Demo. Der Browser greift, nach dem Erlauben dieses Features, auf diverse Informationen zu, wie beispielsweise vom Benutzer definierte Standorte, GPS-Sender (in mobilen Geräten), WLAN/Handy-Netzwerke und natürlich die IP des Benutzers. Anhand dieser Informationen kann die API, je nach Internetprovider und örtlicher DSL-Infrastruktur, die Position, über geografische Koordinaten, welche mit einer Karte abgeglichen werden, mit einer Genauigkeit zwischen 1km bis 100km bestimmen. Bei mobilen Geräten mit GPS-Sender sogar auf einige Meter genau[46]. Wie immer gilt auch hier, nachdenken und dann klicken, da auch dieses Feature Ziel von SE Angriffen sein kann und man auch wieder Opfer eines XSS Angriffs werden kann, sodass der Angreifer diese Informationen abrufen kann, sofern die XSS Lücke in einer vertrauten Webseite war. Basierend auf diesen Informationen kann ein Angreifer dem Opfer z.B. lokal Werbung zukommen lassen, im Falle von mobilen Geräten, die Verhaltensweisen eines Opfers studieren, indem Routen angefertigt werden (da man Positionen cachen kann), die Adresse des Opfers weitergeben oder auch überprüfen, ob ein Opfer im Moment nicht in seiner Wohnung ist, sodass Einbrecher diese Information nutzen können. Aufgegriffen wurde dieses Thema von der Electronic Frontier Foundation in einer Arbeit über Location Privacy[2].

3.10 Weitere Features

Der Vollständigkeit halber sei erwähnt, dass neben den bisher genannten Features auch noch weitere, teils experimentelle, Features geplant sind, wie die *Media Capture API*, *System Information API*, *Speech Input*, *Custom Protocol/-Content Handler* und *Widgets*, auf welche hier allerdings nicht weiter eingegangen wird.

[44]hier ist zu beachten, es kann jede History URL mit replaceState() ersetzt werden, aber nur in eine URL mit der selben Domain wie das aufrufende Script
[45]`http://html5demos.com/geo`
[46]`http://dev.w3.org/geo/api/spec-source.html`

Sicherheits-kritische HTML5 Features	Chrome 19 402 13 bonus points	Internet Explorer 10 319 6 bonus points	Firefox 13 345 9 bonus points
SVG in `text/html`	Yes ✓	Yes ✓	Yes ✓
`canvas` element `audio` element `video` element	Yes ✓	Yes ✓	Yes ✓
`keygen`	Yes ✓	No ✗	No ✗
Session history	Yes ✓	Yes ✓	Yes ✓
Application Cache	Yes ✓	Yes ✓	Yes ✓
Custom scheme handlers	Yes ✓	No ✗	Yes ✓
Custom content handlers	No ✗	No ✗	Yes ✓
Custom search providers	Yes ✓	Yes ✓	Yes ✓
Sandboxed `iframe`	Yes ✓	Yes ✓	No ✗
Geolocation	Yes ✓	Yes ✓	Yes ✓
Cross-document messaging	Yes ✓	Yes ✓	Yes ✓
XMLHttpRequest Level 2	Yes ✓	Yes ✓	Yes ✓
WebSocket	Yes ✓	Yes ✓	Yes ✓
FileSystem API	Yes ✓	No ✗	No ✗
Session Storage	Yes ✓	Yes ✓	Yes ✓
Local Storage	Yes ✓	Yes ✓	Yes ✓
IndexedDB	Yes ✓	Yes ✓	Yes ✓
Web SQL Database	Yes ✓	No ✗	No ✗
Web Workers	Yes ✓	Yes ✓	Yes ✓
Web/Desktop Notifications	Yes ✓	No ✗	No ✗

Abbildung 4: Browservergleich[47]

3.11 Bisheriger Implementierungsstand

Zum Vergleich der Implementierung der obigen Features in den aktuellen Browsern Google's Chrome 19, Microsoft's Internet Explorer 10 und Mozilla's Firefox 13, wird verwiesen auf die Webseite **HTMLTest.com**, welche sich zur Aufgabe gemacht hat, alle HTML5 Features und deren Browserimplementierung übersichtlich und aktuell der Öffentlichkeit zur Verfügung zu stellen. Ein Auszug daraus befindet sich in Abbildung 4. Wie man in der Tabelle erkennt, sind fast alle oben genannten Features in allen verbreiteten Browsern implementiert und bereit, genutzt und ausgenutzt zu werden.

4. ALLGEMEINE VERBESSERUNGSVORSCHLÄGE

Grundsätzlich sollten Benutzer mehr Aufklärung in allen sicherheitsrelevanten Bereichen des Web erfahren. Es ist in vielen Fällen nicht möglich die Balance zwischen automatischer Browsersicherheit und komfortablen Surfens zu erhalten. Deshalb müssen Benutzer im Speziellen auf, vom Browser verlangte Rechte, mit Sachverstand reagieren (Stichwort *Geolocation API* und *Desktop Notification API*). Wie im Verlauf der Arbeit aufgefallen ist, basieren viele Sicherheitslücken im Web auf client-seitigen Scriptsprachen wie JavaScript. Man denke dabei an XSS Lücken, durch welche viele weitere Angriffe möglich sind, aber ohne Scripte nicht möglich wären. Eine POC-Abwehr bösartiger Scripts wäre das

[47]Stand: 22.06.2012 `http://html5test.com/compare/browser/chrome19/ie10/ff13.html`

Browserplugin *NoScript*[48], welches kostenlos für viele Browser zur Verfügung steht. Die Sicherheit dieser Methode hängt am Ende allerdings doch wieder von dem Verhalten des Nutzers ab, da es sehr restriktiv, entweder Scripte erlaubt oder verbietet, was für die Nutzung moderner Webanwendungen sehr unkomfortabel ist. Webentwickler sollten sicherheitskritische HTML5 Features mit besonderer Vorsicht implementieren (Stichwort *Sandboxed Iframes* und *CORS*). Sie sollten XSS-Lücken schließen, indem sie Benutzereingaben korrekt auswerten, beispielsweise mit Whitelists, anstelle von Blacklists und sollten aus den hier genannten neuen Angriffen und Angriffsvektoren verschiedener HTML Tags lernen und ihre Implementierungen anpassen. Die nützliche Web Storage Funktion sollte nicht genutzt werden um sensible Daten zu speichern, ebenso sollten die Sessions eher von Cookies verwaltet werden[17]. Für größere Webanwendungen sollten sich die Entwickler im Vorfeld klar werden, welche Sicherheitsmaßnahmen diese erfordern. Beispielsweise im Blog von Kaazing wird dazu genau beschrieben, wie man die Sicherheit von Web Sockets ausreichend für die eigenen Bedürfnisse ausbaut[9]. HTML5 ist erst in den Kinderschuhen und wird in Zukunft weite Verbreitung finden, welche auf frühen Implementierungen basiert. Diese abzusichern, sollte das Hauptziel der aktuellen Webentwickler sein, welche mit ihren Arbeiten Wegbereiter von HTML5 werden. Ein allgemeiner Verbesserungsvorschlag für Browserhersteller, der auch der Benutzeraufklärung im Bereich Web-Sicherheit dienen würde, wäre die Implementierung von intuitiveren Konfigurationsdialogen. Die leichte Benutzbarkeit von Browsern sollte zwar erhalten bleiben, dennoch müssen die Lücken zwischen Unwissen und Wissen bei Benutzern, im Bereich Web-Sicherheit geschlossen werden. Eine mögliche Lösung wären Browser-Sicherheitsprofile. Diese könnten über Assistenten erstellt werden, welche die Benutzer über relevante Sicherheitsbedenken aufklären, um so, für jeden Benutzer individuell, die optimale Sicherheitseinstellung definieren zu können. Weiterhin sollten Benutzer spezielle Sicherheitsprofile anlegen können und leicht zwischen ihnen wechseln. Diese könnten z.B. sein: Online-Banking, Shopping, Social-Networking, etc. [20].

5. ZUSAMMENFASSUNG

HTML5 ist ein großer Schritt in eine bessere Internet-Zukunft, mit durchdachten Sicherheitskonzepten, aber dennoch kommt es immer noch auf die richtige und sichere Verwendung dieser neuen Features an. Mit der bisherigen Implementierung von HTML5 könnten in der Zukunft viele Angriffsszenarien entstehen, wie sie zum Teil in dieser Arbeit beschrieben wurden. Es ist ein, noch lange nicht abgeschlossener, iterativer Prozess, die HTML5 Features zu verbessern und es ist klar, dass die Sicherheitsspezifikationen noch nicht bei allen Features ausgereift sind, dennoch beschäftigt sich das W3C schon mit HTML6, der Generation nach HTML5[49]. Aus diesem Grund liegt es an den Entwicklern Vorschläge zu unterbreiten, welche den HTML5 Standard sicherer machen können. Das Web, als größtes Einfallstor, wird in Zukunft immer öfter in die Schlagzeilen kommen und schon jetzt beginnt der Wettstreit zwischen Cyberkriminellen und der Sicherheitsindustrie.

[48] `http://noscript.net/`

[49] `http://www.heise.de/newsticker/meldung/Erste-Ideen-fuer-HTML6-903955.html`

6. REFERENZEN

[1] Andlabs.org: *Chrome and Safari Users Open to Stealth HTML5 AppCache Attack.* Blog: `http://blog.andlabs.org/2010/06/chrome-and-safari-users-open-to-stealth.html`,

[2] Andrew J. Blumberg, Peter Eckersley: On Locational Privacy, and How to Avoid Losing it Forever. (2009)

[3] Balduzzi, M. : *New Insights into Clickjacking.* OWASP: `https://www.owasp.org/images/b/b4/OWASP_AppSec_Research_2010_Clickjacking_by_Balduzzi.pdf`, 2010

[4] Eilers, C. : Client Security im Web 2.0 und mit HTML5. In: *WebTech* (2011)

[5] Han, G. : Clickjacking-Angriff auf Webseiten. (2011)

[6] Heiderich, M. : *HTML5 Security Cheat Sheet.* Blog: `http://html5sec.org/`,

[7] Heiderich, M. : The Image that called me. (2011). – `https://www.owasp.org/images/0/03/Mario_Heiderich_OWASP_Sweden_The_image_that_called_me.pdf`

[8] Johns, M. : HTML5-Security Sicherer Umgang mit den neuen Javascript APIs. In: *DuD - Datenschutz und Datensicherheit* (2012)

[9] Kaazing: *WebSocket Security Is Strong.* Blog: `http://blog.kaazing.com/2012/02/28/html5-websocket-security-is-strong/`, 2012

[10] Kotowicz, K. : *HTML5 Something Wicked This Way Comes.* `https://connect.ruhr-uni-bochum.de/p3g2butmrt4/`,

[11] Kuppan, L. : *Autocomplete Demo.* `http://www.andlabs.org/hacks/steal_autofill.html`,

[12] Kuppan, L. : *HTML5 Sicherheitsdemos.* Blog: `http://www.andlabs.org/html5.html`,

[13] Kuppan, L. : Attacking with HTML5. (2010)

[14] Lin-Shung Huang, Eric Y. Chen, Adam Barth, Eric Rescorla & Collin Jackson: Talking to Yourself for Fun and Profit. (2011)

[15] McArdle, R. : HTML5 Overview: A Look at HTML5 Attack Scenarios. (2011)

[16] MDN: *HTTP Access Controll (CORS).* Blog: `https://developer.mozilla.org/en/http_access_control`,

[17] Michael Schmidt, Compass Security AG: HTML5 Web Security. (2011)

[18] Nyman, R. : *postMessage Beispiel.* Blog: `http://robertnyman.com/2010/03/18/postmessage-in-html5-to-send-messages-between\-windows-and-iframes/`,

[19] OWASP: *The OWASP Top 10 Web Application Security Risks.* `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`,

[20] Philippe De Ryck, Lieven Desmet, Pieter Philippaerts, and Frank Piessens, Katholieke Universiteit Leuven: A Security Analysis of Next Generation Web Standards. In: *ENISA* (2012)

[21] Raskin, A. : *Tabnabbing: A New Type of Phishing Attack.* Blog: `http://www.azarask.in/blog/post/a-new-type-of-phishing-attack/`,

[22] Shah, S. : HTML5 Top 10 Threats Stealth Attacks

and Silent Exploits. . –
http://media.blackhat.com/bh-eu-12/shah/
bh-eu-12-Shah_HTML5_Top_10-WP.pdf

[23] Sharp, R. : *HTML5Doctor.com - WebSQL Beispiele.* Blog: http://html5doctor.com/introducing-web-sql-databases/,

[24] Steve Hannax, Eui Chul Richard Shinz, Devdatta Akhawex, Arman Boehmz, Prateek Saxenax & Dawn Song: The Emperors New APIs: On the (In)Secure Usage of New Client-side Primitives. (2010)

[25] Steven Kapinos, Michael Cross & Steven H. Palmer: *Web Application Vulnerabilities: Detect, Exploit, Prevent.* Syngress Media, 2007

[26] W3C: *Cross Domain Resource Sharing.* http://www.w3.org/TR/cors/,

[27] W3C: *Cross-Origin Resource Sharing.* http://www.w3.org/TR/cors/,

[28] W3C: *HTML5 Differences From HTML4.* http://dev.w3.org/html5/html4-differences/,

[29] W3CSchool: *W3C School - SVG.* http://www.w3schools.com/html5/html5_svg.asp,

[30] Wiliam West & S. Monisha Pulimood: Analysis of Privacy and Security in HTML5 Web Storage. (2011)

[31] Zalewski, M. : *The Tangled Web A Guide To Securing Modern Web Applications.* No Starch Press, 2011