HOCHSCHULE DER MEDIEN

Bachelor Thesis

Internet-Security aus Client-Sicht am Beispiel des Internet Explorers und des Firefox: Angriffsmöglichkeiten und Abwehrmaßnahmen

Studiengang: Medieninformatik
Verfasst von: Bastian Zimmermann
Erstprüfer: Prof. Dr. Roland Schmitz
Zweitprüfer: Prof. Walter Kriha
Bearbeitungszeitraum: 27.10.2008 – 26.01.2009

Hochschule der Medien Stuttgart

Kurzfassung

Gegenstand dieser Arbeit ist die clientseitige Benutzerdaten- und Anwendungssicherheit. In Browsern sind viele Sicherheitsmechanismen und -einstellungen vorhanden. Der Browser selbst und dessen verwendetes Betriebssystem sind schützenswert. So soll es einem Angreifer nicht möglich sein, Viren einzuschleusen oder den Browser zum Absturz zu bringen. In wie weit werden aber auch Maßnahmen zum Schutz der verarbeiteten Benutzerdaten, sowie der dargestellten Webseiten ergriffen?

Das Sicherheitsbedürfnis der privaten Daten und Webanwendungen im Internet steigt an. Im Web werden immer wichtigere Informationen weitergegeben und verarbeitet. Dies gilt nicht nur für Daten von Privatpersonen, sondern auch für Unternehmensdaten. Es gibt viele Dokumente zur Absicherung der Serverseite, jedoch verlassen die Daten diese und müssen deshalb auch clientseitig geschützt werden. Zudem darf es nicht möglich sein, dass clientseitig andere Webseiten Einfluss auf die Webanwendung nehmen.

Ziel dieser Arbeit ist das Aufzeigen möglicher clientseitiger Maßnahmen zum Schutz der Daten und Anwendungen. Ferner wird dargelegt, wie sicher die momentan Browser sind. Des Weiteren werden Browsererweiterungen vorgestellt, die die Sicherheit erhöhen. Auf Schutzmaßnahmen für Browser und Betriebssystem wird nicht eingegangen.

Schlagwörter: Datensicherheit, Anwendungssicherheit, GIFAR, Cross-Site Request Forgery, CSRF, Cross-Site Scripting, XSS, Clickjacking, Microsoft Internet Explorer, Mozilla Firefox, Sicherheitseinstellungen, Browsereinstellungen, NoScript, Controle de Scripts

Abstract

This thesis examines client-side user data and application security. There are many security mechanisms and security settings in browsers. To begin with sufficient protection should be provided for the browser itself and the operating system used. In this way an attacker can be prevented from smuggling in a virus or causing the browser to crash. To which extent, however, is action being taken to protect the user data which has been processed and the displayed websites?

The need to secure private data and web applications in internet is increasing. Information of more and more importance gets passed through the web and is processed there. This applies not only for the data of private individuals but also for company data. There is a lot of documentation for securing the server-side, but the data leaves this side and must also be protected on client-side. Furthermore, other websites must not be allowed to exert influence on web applications on the client-side.

The purpose of this paper is to highlight possible measures to protect data and applications. It also introduces browser extensions offering increased security. It does not deal with security measures for browsers or operating systems.

Keywords: data security, application security, GIFAR, cross-site request forgery, CSRF, cross-site scripting, XSS, clickjacking, Microsoft Internet Explorer, Mozilla Firefox, security settings, browser settings, NoScript, Controle de Scripts

Inhaltsverzeichnis

Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript and XML
CSRF	Cross-Site Request Forgery
CSS	Cascading Style Sheets
DOM	Document Object Model
FF	Mozilla Firefox
GIF	Graphics Interchange Format
(X)HTML	(Extensible) Hypertext Markup Language
http(s)	Hypertext Transfer Protocol (Secure)
IE	Microsoft Internet Explorer
JAR	Java Archive
JSP	JavaServer Pages
JVM	Java Virtual Machine
MIME	Multipurpose Internet Mail Extensions
MITM	Man-In-The-Middle
OCSP	Online Certificate Status Protocol
OWASP	Open Web Application Security Project
SSL	Secure Sockets Layer
TLS	Transport Layer Security
URL	Uniform Resource Locator
XML	Extensible Markup Language
XSS	Cross-Site Scripting

Tabellenverzeichnis

Abbildungsverzeichnis

Code-Beispiel-Verzeichnis

1 Einführung und Motivation

Diese Thesis entstand an der Hochschule der Medien Stuttgart (HdM) im Wintersemester 2008/2009 und ist im Bereich der Internet-Sicherheit anzusiedeln. Die Gründe für die Auswahl dieser Thematik liegen in der Entwicklung des Internets. Es findet ständig eine Weiterentwicklung statt, die Einfluss auf den Alltag von Menschen nimmt. Eine Erleichterung wird beispielsweise durch die Beschaffung verschiedener Güter mittels des häuslichen Einkaufs rund um die Uhr erreicht. Bietet eine Firma einen ansprechenden Internetauftritt, führt dies meistens zu einer Verbesserung des Images und damit auch zu einer Umsatzsteigerung. Auch die Nachrichtenübermittlung hat sich immer mehr ins Netz verlagert, weil dadurch eine schnellere, komfortablere und kostengünstigere Kommunikation möglich ist, als bei herkömmlichen Methoden. Durch den erkennbaren Trend, dass immer mehr Anwendungen ins Web verlagert werden, ist zu erwarten, dass auch das Sicherheitsbedürfnis steigt. Je mehr Daten im Internet kursieren, desto lukrativer ist es für Kriminelle diese zu manipulieren oder zu missbrauchen. Besonders interessant sind dabei persönliche Daten, wie Konto- oder Kreditkartennummern und Unternehmensdaten. Die Verlagerung von Anwendungen ins Internet nimmt immer größere Ausmaße an, wodurch die Gefahr des Datenmissbrauchs deutlich steigt. Eine neue Technologie für die Nutzung von Online-Diensten stellt das Cloud Computing dar. Hierbei wird die komplette Serverhardware, auf der alle benötigten Anwendungen installiert sind, von einem Anbieter gestellt. Es ergibt sich daraus die Möglichkeit, alle Programme über einen Rechner zu benutzen, der am Internet angeschlossen ist, ohne dass diese lokal installiert sein müssen. Weil der Browser zur Bedienung dieser Software benötigt wird, stellt er eine sehr wichtige Komponente in einem solchen System dar. Die extremste Form von Cloud Computing besteht darin, dass clientseitig nur noch ein Browser läuft und alle weiteren Programme von der Cloud bereitgestellt werden. Sobald ein Bestandteil des Konzepts eine Schwachstelle aufweist, ist die Gesamtsicherheit bedroht. Dabei können nicht nur Server und Browser einem Angriff zum Opfer fallen, sondern auch Benutzerdaten und Anwendungen.

Während zum Absichern von Servern viele Bücher, Dokumentationen und andere wichtige Hinweise vorhanden sind, lassen sich Informationen zum Schutz des Clients kaum finden. Aus diesem Grund beschäftigt sich meine Thesis mit der Internet-Security aus Client-Sicht. Ich untersuche die Anwendungs- und Benutzerdatensicherheit von Browsern, die sich mit der Sicherheit von Anwendungs- und Benutzerdaten, sowie der Kommunikation zwischen Browser und Webanwendung und deren Laufzeitverhalten befasst, wohingegen ich Angriffe, die direkt den Browser oder das Betriebssystem betreffen, nicht berücksichtige.

Nachfolgend gebe ich einen Überblick, wo bereits Benutzerdaten- und Anwendungssicherheit nötig ist. Ein offensichtliches Beispiel hierfür ist Online-Banking. In diesem Fall können nicht nur Zugangsdaten gestohlen werden, sondern auch Transaktionen erfolgen, auch ohne Kenntnis von Benutzernamen und Passwort. Durch Angriffe auf Webmailer kann der Zugriff auf das Postfach des Opfers bewerkstelligt werden. So ist es möglich, dessen Mails zu lesen und zu löschen. Besteht keine ausreichende Sicherheit, können Profile in Online-Communities und Foren manipuliert werden, sowie Beiträge im Namen des Benutzers verfasst werden. Weist die Weboberfläche einer Netzwerkkomponente, wie z. B. einem Router, Schwachstellen auf, können deren Einstellungen abgeändert werden. Wenn man sich an den Meinungen der Anbieter von Cloud-Computing-Systemen orientiert, brauchen Unternehmen keine eigene Serverinfrastruktur mehr, sondern sollen stattdessen ihre Anwendungen in die Cloud verschieben (Diercks, 2008; Wilkens, 2008). Somit soll

sich der größte Teil der Betriebskosten für große Anwendungen sparen lassen. Für Unternehmen ist ihr Wissen das höchste Gut und deshalb werden Unternehmen solche Verlagerungen nur durchführen, wenn sie sicher gehen können, dass auch die Daten sicher aufgehoben sind. Die meisten Unternehmen stehen dem eher skeptisch gegenüber. Durch die vielen Meldungen über Manipulationen von Webanwendungen und über Klauen von Benutzerdaten wird diese Meinung unterstützt.

Aber warum sind Webanwendungen so unsicher? Liegt dies nur daran, dass die Programmierer, die solche entwickeln, Fehler machen? Wenn man an viele konservative Angriffe denkt, wie an Umgehen von Autorisierungsmechanismen, die mit JavaScript umgesetzt wurden, oder an einfaches Abändern des Quelltextes, um Zugriff auf andere Profile zu erhalten, mag dies der Fall sein. Aber viele neue Angriffe setzen auf höheren Ebenen an. So gibt es Angriffe, die autorisierte Verbindungen des Browsers verwenden oder Klicks im Browser geschickt umlenken. Bei solchen Angriffen hat die Webanwendung kaum eine Chance. Deshalb trifft die Programmierer derselben auch keine oder nur eine geringe Schuld bei solchen Problemen. Der Ursprung der Probleme liegt bei solchen Attacken zumeist im Browser. Man hört immer wieder, dass dieser oder jener Browser sicherer sei als die anderen. Hierbei werden allerdings nicht die Anwendungs- und Benutzersicherheit verglichen, sondern die Schwachstellen der Browser, über die beispielsweise Trojaner auf dem Client installiert werden können. Sicherlich ist diese Art von Sicherheit auch wichtig. Die Browserhersteller interessieren sich allerdings vergleichsweise wenig für andere Sicherheitsbereiche.

Die Gründe hierfür liegen in der Entstehung des Internets. Anfangs wurde es nur für wissenschaftliche Zwecke verwendet, später kamen Webpräsenzen auf, bei denen man sich möglichst gut darstellen wollte. Wichtige Benutzer- und Anwendungsdaten gingen hierbei kaum übers Netz. Mit dem Erlangen eines Benutzernamens und des dazugehörigen Passworts konnte man Seiten verunstalten oder sich unbefugten Zugriff auf einen Mitgliederbereich verschaffen. Einen großen wirtschaftlichen Schaden anzurichten war kaum möglich. Nach und nach hielt die Kommerzialisierung Einzug ins Internet. Mit Webshops und Online-Banking stiegen auch die Sicherheitsbedürfnisse. Einige rieten: „Benutzt SSL und ihr seid auf der sicheren Seite.“ Doch auch SSL hat seine Grenzen, nämlich dann, wenn der Angreifer es schafft, dass er authentifizierte Verbindungen mitbenutzen kann. Nachdem sich die Entwickler von Webanwendungen immer ausgeklügeltere Sicherheitsmechanismen überlegt und die Angreifer hingegen immer durch neuere Attacken nachgezogen haben, ist es nun an der Zeit, dass die Browserhersteller Verantwortung zeigen und die Sicherheit von Webanwendungen unterstützen.

Inwiefern sie dies bereits tun, wird in dieser Arbeit anhand der beiden beliebtesten Browser Internet Explorer und Firefox genauer analysiert. Es wird verdeutlicht, welche Unterstützung der Benutzerdaten- und Anwendungssicherheit die Browser bereits mit Standardeinstellungen bieten. Des Weiteren werden Möglichkeiten zur Erhöhung des Schutzes aufgezeigt. Diese Erklärungen sind auch für Laien verständlich. Außerdem werden Schutzmaßnahmen erläutert, die es in Browsern noch nicht gibt, die aber implementiert werden sollten.

Zunächst sind im ersten Kapitel vier ausgewählte Angriffsmöglichkeiten detailliert erklärt. Ich entschied mich für die beiden sehr weit verbreiteten und bereits länger bekannten Angriffe Cross-Site-Scripting (XSS) und Cross-Site Request Forgery (CSRF). Daraufhin folgen zwei neuere Angriffe, die Ende letzten Jahres Aufsehen erregten. Dies ist zum einen GIFAR, eine Technik um einen Webserver ein JAR-Archiv unterzuschieben, und zum anderen Clickjacking, eine Angriffsart, die Klicks im Browser

umleitet. Nachdem man so einen Einblick in mögliche Attacken auf Benutzerdaten und Anwendungen erhalten hat, bekommt man im nächsten Kapitel einen Überblick über mögliche Abwehrmaßnahmen. Zunächst wird die Problematik des Daten- und Anwendungsschutzes allgemein beschrieben. Hierin werden die Probleme und Ursachen, die Angriffe begünstigen, dargestellt. Danach werden Schritte beschrieben, wie man die Missstände beheben kann. Zunächst werden allgemeine Mittel zum Schutz der Daten aufgezählt. Danach werden spezielle Filter zur Steigerung der Sicherheit erklärt. Der erste befasst sich mit der Erhöhung der Sicherheit von JavaScript, der zweite mit der Überprüfung von Anfragen an Webserver zur Verhinderung ungewollter oder verdächtiger Requests. Daraufhin folgt das Kapitel über die Einstellungen der Browser. Anfangs wird hierin die verwendete Testmethodik beschrieben. So erhält man einen Einblick, welche Funktionen der Browser überprüft wurden und wie dies geschehen ist. Daraufhin folgen die Testergebnisse, sowie die Beschreibungen der Sicherheitseinstellungen der Browser. Vorneweg wird erläutert, in welcher Weise der sicherheitsrelevante Umgang der Browser mit verschiedenen Seiten erfolgt. Die Testergebnisse werden entweder der Kategorie „negative Ergebnisse“ oder „positive Ergebnisse“ zugeordnet. Welche Auswirkungen die einzelnen Ergebnisse hervorrufen, wird bereits bei der Testmethodik erklärt. Im Anschluss werden die möglichen Sicherheitseinstellungen der Browser erläutert. Hierbei sind nur Einstellungen erfasst, die über die Benutzeroberfläche des Browsers zugänglich sind und nicht etwa solche, bei denen Einträge in die Windows-Registrierungsdatenbank oder Konfigurationsdateien nötig sind. Die Einstellungen werden leicht verständlich beschrieben und von Bildern unterstützt. Um die Sicherheit von Browsern weiter zu verbessern, gibt es Sicherheitsplugins. Plugins sind in diesem Falle Browsererweiterungen, die durch ihre Funktionen das Verhalten des Browsers ändern. Als Beispiele für Sicherheitsplugins wählte ich „Controle de Scripts“ und „NoScript“ für den Firefox aus. Diese sind in einem separaten Kapitel erklärt. Zusammenfassend ziehe ich im Fazit ein Resümee über den derzeitigen Stand der Benutzerdaten- und Anwendungssicherheit. Abschließend zeige ich im Ausblick Möglichkeiten zur Verbesserungen der Daten- und Anwendungssicherheit auf.

2 Angriffsmöglichkeiten

2.1 Cross-Site-Scripting (XSS)

Cross-Site-Scripting ist heutzutage wohl die bekannteste Angriffstechnik auf Webseiten. Früher wurde hierfür auch die Abkürzung CSS verwendet. Dies führte allerdings zu Verwirrung, da CSS im Web-Umfeld eigentlich die Abkürzung für „Cascading Style Sheets“ ist. Deshalb wurde die Abkürzung XSS, wobei das X für ein Kreuz („Cross“) steht, eingeführt. Fast jeder, der sich mit der Sicherheit in Web-Anwendungen befasst, hat zumindest schon einmal von XSS gehört. Aber was ist XSS eigentlich? Dies werde ich nun erläutern, während ich folgende Quellen zur Unterstützung heranziehe: (Computerwoche, 2007; Huseby, 2004; OWASP Foundation, 2008).

2.1.1 Allgemein

XSS ist eine Angriffstechnik, bei der der Angreifer versucht, interpretierte Zeichenketten in eine Web-Anwendung einzuschleusen. Dies nennt man auch Interpreter Injection, da der Angriff erst durch die Verarbeitung der Zeichenketten durch einen Interpreter ausgeführt wird. Hauptsächlich wird dabei JavaScript gemischt mit HTML verwendet. Dies ist aber nicht zwingend nötig, auch XSS-Angriffe auf Basis von Flash oder anderer aktiver Inhalte sind möglich. Das Gefährliche hierbei ist, dass aktive Inhalte im Kontext der Seite aufgerufen werden, von der sie stammen. Da bei XSS ein eingeschleustes JavaScript aber scheinbar vom vertrauenswürdigen Server stammt, hat es Zugriff auf alle Daten und Variablen, die auch andere JavaScripts der Anwendung haben, insbesondere auch auf das Document Object Model (DOM). Man unterscheidet drei verschiedene Typen von XSS-Angriffen.

2.1.2 XSS-Typen

Bei persistenten Angriffen wird die bösartige Zeichenfolge vom Server dauerhaft gespeichert, z. B. in einer Datenbank. Dies ist beispielsweise bei einem verwundbaren Gästebuch, Forum oder einer Profilseite in einem sozialen Netzwerk denkbar. Der Fehler liegt hierbei im jeweiligen Code, der auf dem Server ausgeführt wird.

Bei nicht-persistenten oder reflektierten Angriffen wird die unheilvolle Zeichenfolge ggf. in veränderter Form an den Client, meist einem Browser mit Unterstützung von aktiven Inhalten, weitergegeben und dort verarbeitet. Hierzu muss das Opfer meist einen präparierten Link aufrufen. Der Fehler liegt wiederum beim Server, der so z. B. HTML-Tags oder JavaScript des Angreifers einbettet. Reflektierte Angriffe sind beispielsweise auf Suchseiten, Fehlerseiten, Anmeldeseiten und Startseiten denkbar, bei denen benutzergenerierte Eingaben wieder angezeigt werden.

Die dritte Form von XSS ist der clientseitige oder DOM-basierte Angriff. Die gefährliche Zeichenfolge muss hierzu vom Client interpretiert werden. Da Web-Anwendungen meist JavaScript einsetzen, wird der Schadcode bei dieser Attacke in einem vom Script der Anwendung verarbeiteten URL-Parameter untergebracht. Allerdings muss auch zu diesem Zweck der Client einem präparierten Link folgen.

2.1.3 Ziele von XSS

Die Ziele des Angreifers können verschiedenster Art sein. Manche Angreifer versuchen „nur“ die Darstellung einer Web-Anwendung zu verunstalten. Dies kann dem Ruf des Webseiten-Betreibers schaden. Andere Angriffe versuchen die Zugriffskontrolle bzw. Anmeldung der Anwendung zu umgehen. Aber auch das absichtliche Verursachen von Anwendungsfehlern zum Gewinnen neuer Informationen kann ein Wunsch des Hackers sein. Außerdem kann der Eindringling versuchen, sensible Daten zu lesen oder zu verändern. Eines der beliebtesten Ziele ist allerdings das Stehlen des

Cookies des Anwenders. So wird der Angreifer meist von der Anwendung als normaler Benutzer, nämlich als der, dessen Cookie er gestohlen hat, identifiziert und kann Aktionen im Namen dieses Anwenders durchführen. Je höher die Privilegien dieses Benutzers sind, desto höher ist auch der potentielle Schaden. Ein weiterer Plan des Angreifers kann es sein, betrügerische Inhalte zu präsentieren, um Opfer auf andere Seiten zu locken. Da durch XSS sogar Weiterleitungen des Opfers veranlasst werden können, muss der Benutzer nicht einmal aktiv auf einen Link klicken, um auf eine vom Hacker gewollte Seite geleitet zu werden. So kann der Angreifer einen Trojaner installieren, Informationen abfischen oder eine Geschäftsschädigung erreichen.

2.1.4 Persistentes XSS

Die schlimmsten Auswirkungen hat persistentes XSS, da jeder Benutzer, der die manipulierte Seite aufruft, von dem Angriff betroffen ist. Der Anwender muss hierzu keine präparierte URL anklicken, sondern einfach nur eine Seite aufrufen, die er vielleicht schon oft besucht hat. Hierbei sind zwei Dinge von großer Bedeutung: Die Zahl der Opfer ist wesentlich größer als bei den anderen Angriffsarten und die Attacke ist für das Opfer nicht erkennbar, da sie schon stattgefunden hat, als der Angreifer die manipulierte Anfrage an den Server stellte. Deshalb wird eine verseuchte Seite zurückgeliefert. Für solch einen Angriff ist eine fehlerhafte Eingabevalidierung der Web-Anwendung verantwortlich. Außerdem muss es dem Benutzer möglich sein, Eingaben zu tätigen, die gespeichert werden.

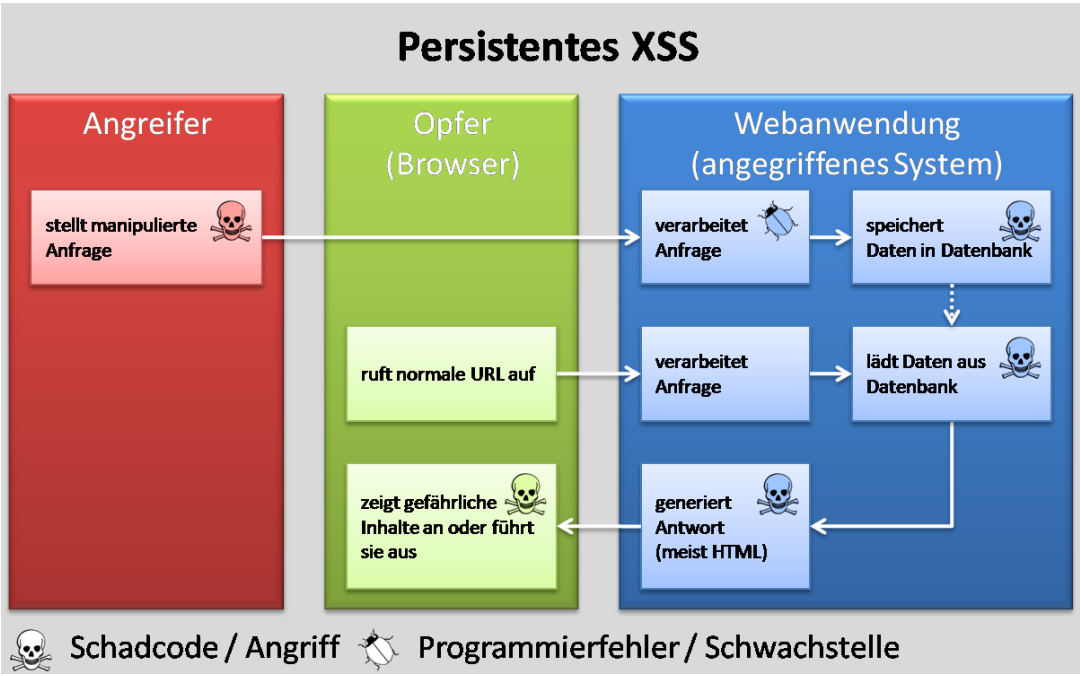

Abbildung 2-1: Persistentes XSS (Rütten, et al., 2007)

2.1.4.1 Beispiel: Gästebuch

Eine solche Anwendung kann beispielsweise ein Gästebuch sein, das keine Benutzereingaben filtert. Der Quelltext der Seite könnte so aussehen:

```
…
<P>
      <B>Hans Meier schrieb:</B><BR />
      Super tolle Seite!
</P>
…
<FORM action="new_guestbook_entry">
<P>
      Ihr Name: <INPUT type="text" name="my_name" /><BR/>
      Ihr Eintrag: <TEXTAREA name="my_text"></TEXTAREA><BR/>
      <INPUT type="submit" value="abschicken" /><BR />
</P>
</FORM>
…
```

Code-Beispiel 2-1: Gästebuch: HTML-Beispiel – gesamt

Der Quellcode der Web-Anwendung, nehmen wir an, sie ist in Java geschrieben, könnte so aussehen:

```
…
String name = request.getParameter("my_name");
String text = request.getParameter("my_text");

if(name != null && text != null){
      Guestbook gb = Guestbook.getGuestbook();
      gb.addEntry(new GuestbookEntry(name, text));
}
…
```

Code-Beispiel 2-2: Gästebuch: Java-Beispiel – Servlet

```
…
<% Guestbook gb = Guestbook.getGuestbook();
GuestbookEntry[] entries = gb.getEntries();
for (GuestbookEntry guestbookEntry : entries) {%>
      <P>
            <B><%= guestbookEntry.getName()+" schrieb:" %></B><BR />
            <%= guestbookEntry.getEntry() %>
      </P>
<% } %>
…
```

Code-Beispiel 2-3: Gästebuch: Java-Beispiel – JSP

Wenn nun ein Benutzer die Angaben „Max Mueller“ bei Name und „Finde ich auch!“ bei Eintrag macht, erzeugt der Webserver hierfür Folgendes:

```
…
<P>
      <B>Max Mueller schrieb:</B><BR />
      Finde ich auch!
</P>
…
```

Code-Beispiel 2-4: Gästebuch: HTML-Beispiel – normaler Eintrag

Wenn nun aber ein Angreifer die Angaben „Hacker“ bei Name und „<SCRIPT>document.location='http://badsite.example.com'</SCRIPT>“ bei Eintrag macht, liefert der Webserver hierfür Folgendes zurück:

```
…
<P>
      <B>Hacker schrieb:</B><BR />
      <SCRIPT>document.location='http://badsite.example.com'</SCRIPT>
</P>
…
```

Code-Beispiel 2-5: Gästebuch: HTML-Beispiel – Angriff

Dies veranlasst den Browser des Opfers bei aktiviertem JavaScript dazu, die Seite „http://badsite.example.com“ aufzurufen.

2.1.4.2 Abwehrmaßnahmen für persistentes XSS

Da persistentes XSS von Client-Seite aus nicht erkannt werden kann, ist die einzig effektive Maßnahme die Eingabevalidierung seitens des Servers. Hierauf wird allerdings nicht näher eingegangen, da dies nicht zum Thema dieser Thesis gehört.

Kann man also clientseitig gar nichts tun?

Doch, man kann, allerdings erfordert es einen hohen Aufwand. Der Erfolg der aufgeführten Mittel ist nicht durchschlagend, da hierdurch Einschränkungen beim Nutzen bestimmter Web-Anwendungen entstehen können. Außerdem können die Angriffe i. d. R. nicht direkt abgefangen, sondern lediglich ihre Folgen gemildert werden.

Eine rabiate Möglichkeit wäre JavaScript und andere aktive Inhalte zu deaktivieren. Dies bietet allerdings auch keine vollständige Sicherheit, da durch geschicktes Verunstalten einer Webseite ohne JavaScript auch Probleme entstehen können. So kann dem Anwender z. B. ein gefälschtes Anmeldeformular untergeschoben werden. Außerdem benötigen viele Webseiten JavaScript, um korrekt zu funktionieren. Deshalb steht diese Methode bei den meisten Benutzern nicht zur Debatte.

Eine weitere mögliche Einschränkung bei JavaScript wäre ein Filter, der gefährliche Befehle abweist. So sollte das JavaScript einer Anwendung beispielsweise nicht auf das Cookie und auf document.location zugreifen. Hierdurch wird der größte Teil von Cookie-Klau-Attacken durch XSS zunichte gemacht und auch gefährliche Weiterleitungen werden erschwert. Allerdings werden leider auch diese Funktionen von etlichen Web-Anwendungen genutzt und deshalb ist eine solche Einschränkung nicht überall möglich. Eine genauere Übersicht zu JavaScript-Filtern befindet sich in Abschnitt [JavaScript-Filter – Seite 42].

2.1.5 Reflektiertes XSS

Bei reflektiertem XSS wird dem Client ein infizierter Code übergeben, der von ihm selbst stammt. Dies bedeutet, der Client ruft eine Webseite auf und übergibt ihr Werte, z. B. durch URL-Parameter, die sie zu einem ungewünschten Verhalten veranlassen. Der eigentliche Angriff erfolgt dann aber auf dem Client und läuft wie beim persistenten XSS ab. Im Gegensatz zu persistentem XSS erreicht der Angriff nur einen einzelnen Nutzer, die Auswirkungen treten allerdings sofort ein. Hierzu muss der Client jedoch eine präparierte URL aufrufen. Dies kann zumeist durch Social Engineering erreicht werden. So werden präparierte URLs oft per Mail verschickt, wobei die E-Mails entweder Interesse erwecken („Nacktbilder von …“), Mitleid hervorrufen („Der / die kleine XY braucht dringend …“) oder Angst auslösen („Daten gingen verloren …“, „Ihr Benutzerkonto wird gelöscht …“) sollen. Es sind aber auch andere Ansätze denkbar, z. B. Posten eines Links in einem Forum. Für den Angriff sind zwei Faktoren verantwortlich. Zum einen müssen der Anwender und damit dessen Browser die

manipulierte URL aufrufen, zum anderen muss die Web-Anwendung Benutzereingaben zurückliefern, die nicht genau genug geprüft werden.

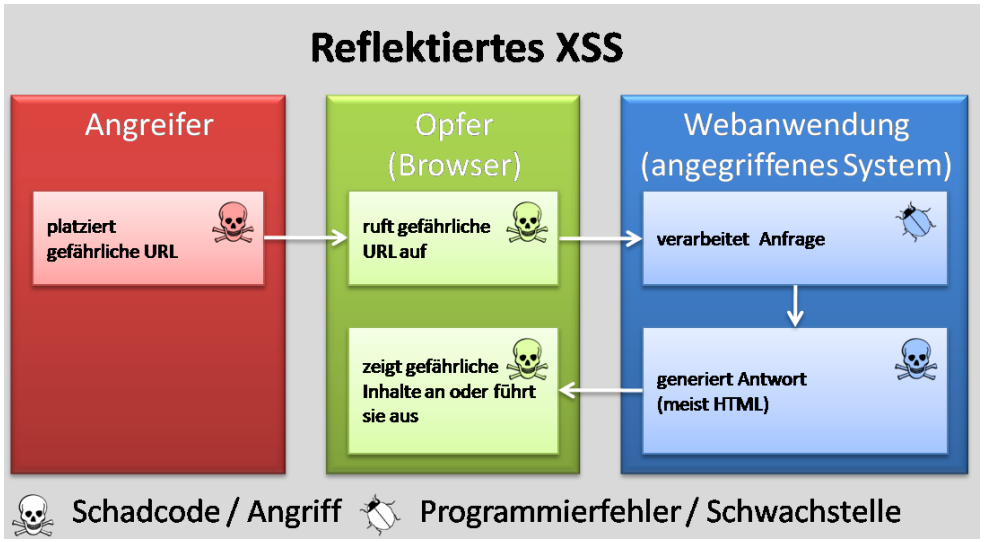

Abbildung 2-2: Reflektiertes XSS (Rütten, et al., 2007)

2.1.5.1 Beispiel: Suchseite

Als Beispiel für eine solche Anwendung dient eine Suchseite einer Anwendung, die den Suchbegriff des Benutzers als Text wiedergibt. Der Quelltext der Seite könnte so aussehen:

```
…
<P>
      Sie suchten nach: <B>abc</B><BR />
</P>
…
<FORM action="Search">
<P>
      Suchen nach:
      <INPUT type="text" name="q" value="abc"/><BR/>
      <INPUT type="submit" value="abschicken" /><BR />
</P>
</FORM>
…
```

Code-Beispiel 2-6: Suche: HTML-Beispiel – gesamt

Der Quellcode der Web-Anwendung, nehmen wir an, sie ist in Java geschrieben, könnte so aussehen:

```
String query = request.getParameter("q");
if(query != null){
      String sr = Find.getResultAsHtml(query);
      if (sr==null){
            sr = "<H1>Leider nichts gefunden</H1>";
      }
      request.setAttribute("sr", sr);
}
```

Code-Beispiel 2-7: Suche: Java-Beispiel – Servlet

```
<% String htmlSearchResult = (String)request.getAttribute("sr");
      String query = request.getParameter("q");
      if(query == null){query="";}
      if(htmlSearchResult != null){
%>
      <P>
            Sie suchten nach: <B><%= query %></B><BR />
      </P>
      <BR />
      <%=htmlSearchResult %>
<% } %>
      <BR />
      <FORM action="Search">
      <P>
            Suchen nach:
            <INPUT type="text" name="q" value="<%= query %>"/><BR />
            <INPUT type="submit" value="abschicken" /><BR />
      </P>
      </FORM>
```

Code-Beispiel 2-8: Suche: Java-Beispiel – JSP

Wenn nun ein Benutzer die URL „www.example.com/search.jsp?q=123“ aufruft, wird vom Webserver hierfür Folgendes erzeugt:

```
…
<P>
      Sie suchten nach: <B>123</B><BR />
</P>
…
```

Code-Beispiel 2-9: Suche: HTML-Beispiel – normaler Eintrag

Wenn nun aber ein Anwender die präparierte URL „www.example.com/search.jsp?q= <SCRIPT>document.location='http://badsite.example.com'</SCRIPT>“ aufruft, liefert der Webserver Folgendes zurück:

```
…
<P>
      Sie suchten nach: <B><SCRIPT>
document.location='http://badsite.example.com'</SCRIPT></B><BR />
</P>
…
```

Code-Beispiel 2-10: Suche: HTML-Beispiel – Angriff

Auch hier veranlasst dies den Browser des Opfers bei aktiviertem JavaScript dazu, die Seite „http://badsite.example.com“ aufzurufen.

2.1.5.2 Abwehrmaßnahmen für reflektiertes XSS

Da bei reflektiertem XSS der manipulative Aufruf von Client-Seite aus getätigt wird, hat dieser nun bessere Möglichkeiten zur Erkennung eines solchen Angriffs. Die Gefahr liegt i. d. R. in übergebenen URL-Parametern. Deshalb ist eine clientseitige Filterung der URL möglich. Es können allerdings auch Formulare abgesendet werden, die dann ebenfalls gefiltert werden müssen. Empfehlungen für eine solche Filterung befinden sich in Abschnitt [Request-Filter (URL-Filter und Form-Filter) – Seite 44]. Solche Filter bieten allerdings keinen vollständigen Schutz, da sie nicht auf die Web-Anwendung zugeschnitten werden können. Kreative Angreifer finden deshalb bei anfälligen Seiten trotzdem

Mittel und Wege, XSS-Attacken einzuschleusen. Werden Request-Filter zusammen mit Filtern gegen aktive Inhalte (siehe auch [JavaScript-Filter – Seite 42]) verwendet, die auch Auswirkungen persistenter XSS-Angriffe abschwächen, kann die Sicherheit weiter erhöht werden.

2.1.6 DOM-basiertes XSS

Bei DOM-basiertem XSS wird dem Client, ähnlich wie beim reflektierten XSS, eine vergiftete URL untergeschoben. Allerdings behandelt der Server die manipulierten URL-Parameter nicht, sondern es wird auf diese vom Client aus zugegriffen, z. B. über JavaScript. Auch dieser Angriffstyp erreicht nur den Anwender, der die infizierte URL aufruft. Auch er ist sofort vom Angriff betroffen. Die Methoden, den Benutzer zum Klicken auf die URL zu verleiten, sind die gleichen wie bei reflektiertem XSS, da das gewünschte Nutzerverhalten, nämlich Anklicken einer manipulierten URL, das gleiche ist. Zum Gelingen des Angriffs ist es nötig, dass der Client die veränderten URL-Parameter verarbeitet. Hierzu ist i. d. R. JavaScript verantwortlich.

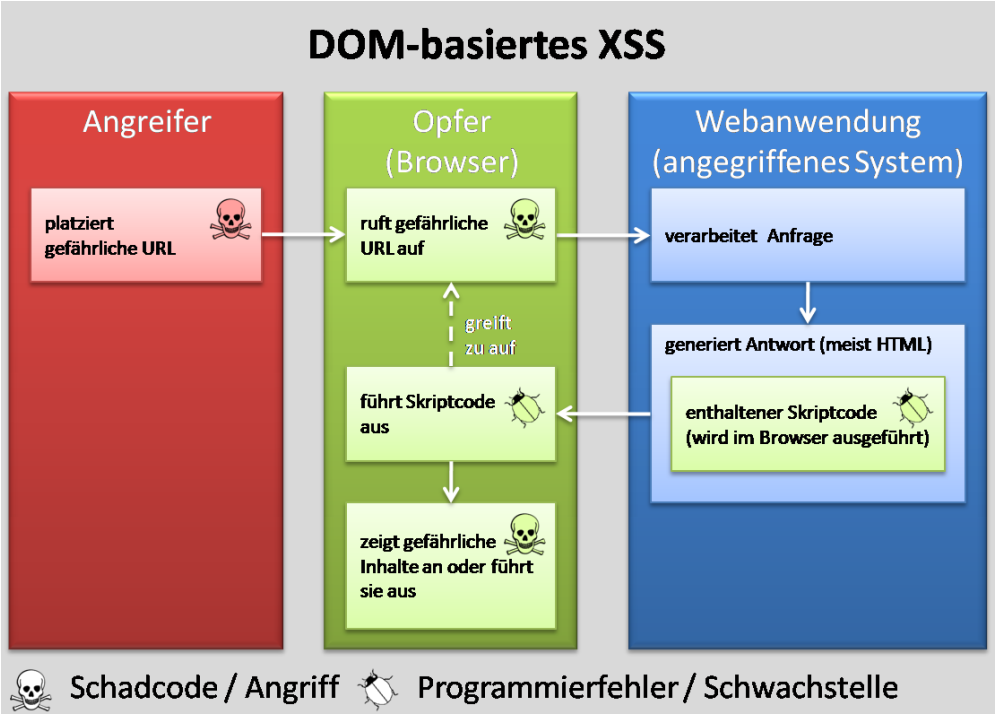

Abbildung 2-3: DOM-basiertes XSS (Rütten, et al., 2007)

2.1.6.1 *Beispiel: Begrüßung*

Ein Beispiel für eine Anwendung, die URL-Parameter verarbeitet, ist eine Seite, die einen Nutzer mit dem Namen begrüßt. Der Quelltext der Seite zum Eingeben des Namens könnte so aussehen:

```
…
<FORM action="welcome.html" method="get">
      Dein Name: <INPUT type="text" name="my_name" /><BR/>
      <INPUT type="submit" value="abschicken" /><BR />
</FORM>
…
```

Code-Beispiel 2-11: Begrüßung: HTML-Beispiel – Eingabe

Der Quellcode der Seite, die die Begrüßung enthält, könnte so aussehen:

```
…
<P>
      Hallo
      <SCRIPT>document.write(getURLParam("my_name"));</SCRIPT>
</P>
…
```

Code-Beispiel 2-12: Begrüßung: HTML-Beispiel – Ausgabe

Wenn nun ein Benutzer die URL „www.example.com/welcome.html?my_name=Michael“ aufruft, wird vom **Browser** hierfür Folgendes erzeugt:

```
…
<P>
      Hallo
      Michael
</P>
…
```

Code-Beispiel 2-13: Begrüßung: HTML-Beispiel – normaler Eintrag

Wenn nun aber ein Anwender die präparierte URL „www.example.com/search.jsp?my_name= <SCRIPT>document.location='http://badsite.example.com'</SCRIPT>“ aufruft, liefert der **Browser** Folgendes zurück:

```
…
<P>
      Hallo
      <SCRIPT>document.location='http://badsite.example.com'</SCRIPT>
</P>
…
```

Code-Beispiel 2-14: Begrüßung: HTML-Beispiel – Angriff

Dies veranlasst den Browser die Seite „http://badsite.example.com“ aufzurufen.

2.1.6.2 Abwehrmaßnahmen für DOM-basiertes XSS

Die Möglichkeiten der Erkennung von DOM-basiertem XSS unterscheiden sich normalerweise nicht von denen zum Erkennen von reflektiertem XSS. Die Gefahr geht im Allgemeinen bei beiden Angriffsarten von den abgeänderten URL-Parametern aus. Deshalb ist eine clientseitige Filterung, wie in [Request-Filter (URL-Filter und Form-Filter) – Seite 44] beschrieben, möglich. Außerdem sollte auch die Web-Anwendung eine solche Filterung durchführen, da diese striktere, auf die Anwendung zugeschnittene Regeln anwenden kann. In diesem Fall ist die Web-Anwendung nicht für XSS-Attacken angreifbar. Außerdem sollten zusätzlich möglichst JavaScript-Filter (siehe auch [JavaScript-Filter – Seite 42]) verwendet werden.

Theoretisch kann man beim DOM-basierten XSS auf Basis von JavaScript die Filterung von Parametern genau auf die Anwendung zuschneiden, da der Quelltext der Webanwendung vorliegt. Dieser Aufwand ist allerdings unüberschaubar groß, da solche Quelltexte schnell einige hundert Zeilen beinhalten können. Einen genau zugeschnitten JavaScript-Filter zu entwerfen ist daher unzumutbar, da dies für jede genutzte Seite gemacht werden müsste. Dennoch ist solch ein Ansatz nicht ganz von der Hand zu weisen. Mehr hierzu in Abschnitt [JavaScript-Filter – Seite 42].

2.1.7 Schwachstellen

Wenn man von XSS-Angriffen hört, denkt man meist an „böses“ JavaScript. Aber was ist „böses“ JavaScript und ist JavaScript die einzige Möglichkeit für XSS-Attacken?

Wo die größten Gefahren bei JavaScript liegen, wird folgender Abschnitt zeigen.

2.1.7.1 Beachtenswerte JavaScript-Funktionen

JavaScript findet im Webumfeld sehr häufig Verwendung. Sehr viele dynamische Web-Anwendungen setzen clientseitig auf JavaScript. Vor allem AJAX-Anwendungen (Asynchronous JavaScript and XML) kommen ohne diese Skriptsprache nicht mehr aus. An sich ist es auch nicht verwerflich, den Client mit Funktionen zu erweitern, doch da diese „Erweiterungen“ nicht unter der Kontrolle des jeweiligen Anwenders liegen, können sie gefährlich sein. Ohne JavaScript ist das Benutzen des Internets meist sicherer, dafür muss ein Benutzer aber auch mit Einschränkungen leben. Auf manchen Seiten fehlt nur die eine oder andere Funktion, um die Benutzbarkeit oder Darstellung zu verbessern, bei anderen vielleicht Funktionen, die man zwar vermisst, aber nicht unbedingt braucht. Einige Webseiten jedoch sind ohne JavaScript nahezu funktionsunfähig. Da man zumeist also auf JavaScript angewiesen ist, sollte man zumindest in Erwägung ziehen, einige Einschränkungen vorzunehmen, falls dies möglich ist.

Im Folgenden werden einzelne JavaScript-Objekte, -Eigenschaften und -Methoden aufgeführt und es wird erklärt, warum diese gefährlich werden können:

- document: Das document Objekt repräsentiert die gesamte Seite. Im Prinzip können alle Funktionen zum Bearbeiten von Elementen gefährlich sein. Da der Angreifer beim XSS den Inhalt jedoch sowieso ändert, werden diese Methoden nicht weiter beachtet, da die Änderung ggf. auch schon vorher erfolgt sein kann (XSS-Attacke auf dem Server) oder vom Client anderweitig leichter erkannt werden kann (clientseitiges XSS). Allerdings werden Eigenschaften betrachtet, auf die es ein Angreifer abgesehen haben könnte.
    - document.URL: Die Eigenschaft URL des Objekts document liefert die vollständige URL des aktuellen Dokuments. Der Zugriff darauf kann nur lesend erfolgen. Warum kann dies trotzdem gefährlich sein? Hat man auf einen manipulierten Link geklickt, enthält die URL die vom Angreifer festgelegten Werte. Wenn eine Webanwendung diese verarbeitet, kann dies schwerwiegende Folgen haben. Die Daten sollte eine Anwendung immer aus der http-Antwort erhalten. Deshalb sollte URL nicht benutzt werden und ist eine Eigenschaft, auf die man achten sollte.
    - document.URLUnencoded: Die Eigenschaft URLUnencoded liefert die decodierte URL des aktuellen Dokuments zurück. Im Übrigen entspricht sie der Eigenschaft document.URL. Diese Eigenschaft ist im Firefox nicht verfügbar und funktioniert im Internet Explorer nicht korrekt.
    - document.cookie: Mit der Eigenschaft cookie kann auf alle Cookies, die zum jeweiligen Dokument gehören, zugegriffen werden. Es können auch neue Cookies gesetzt werden. Gerade das Ausspionieren des Cookies ist eines der wichtigsten Ziele von Angreifern, da sie sich mit dem Cookie bei der Webanwendung meist als der eingeloggte Benutzer ausgeben können.
    - document.referrer: In der Eigenschaft referrer des document Objekts wird die URL der aufrufenden Seite gespeichert, woraus der Angreifer möglicherweise Informationen gewinnen kann. Allerdings ist die Gefährdung, die vom Lesen dieser

Eigenschaft ausgeht, nicht so groß. Dem Webserver steht ein Referrer-Header zur Verfügung, den die Webanwendung nutzen sollte, falls sie diese Information verarbeitet.

- location: Das location Objekt repräsentiert die vollständige URL der aktuellen Seite. Im Gegensatz zur Eigenschaft URL von document ist location aber auch schreibbar! Dies bedeutet, ein Angreifer kann durch Setzen dieses Objekts den Browser des Benutzers dazu veranlassen, eine beliebige URL aufzurufen. Alle Eigenschaften von location sind lesbar und schreibbar. Das location Objekt lässt sich indirekt über document, window oder frame beziehungsweise direkt benutzen.
    - location.href: Die Eigenschaft href des Objekts location ist eigentlich die korrekte Adresse, um die vollständige URL zu lesen oder zu schreiben. Allerdings kann dies, wie bereits erwähnt, auch direkt über das Setzen von location erfolgen. Beim Lesen und Schreiben unterscheidet sich location.href nicht von location. Allerdings bietet location noch weitere Eigenschaften und Methoden, location.href jedoch nicht.
    - location.protocol: Die Eigenschaft protocol repräsentiert das verwendete Protokoll.
    - location.host: Bei der Eigenschaft host bekommt man den Servernamen oder, falls die Seite mit einer IP aufgerufen wurde, diese mit dem verwendeten Port zurück.
    - location.hostname: Wenn nur der Servername oder die IP benötigt wird, kann man auf die Eigenschaft hostname zugreifen.
    - location.port: Der Port lässt sich durch die Eigenschaft port bestimmen.
    - pathname: Die Eigenschaft pathname liefert die Pfadangabe (inkl. Dateiname) zum angeforderten Dokument.
    - search: Diese Eigenschaft liefert die Zeichenkette, die alle Parameter mit ihren Werten darstellt. Sie kann leer sein, muss ansonsten mit einem Fragezeichen beginnen. Parameter werden von ihren Werten durch ein „=“ getrennt und sind untereinander durch „&“ getrennt. Spezielle Zeichen wie „=“, „&“, „?“, „#“, usw. werden umcodiert (siehe hierzu auch [Request-Filter (URL-Filter und Form-Filter) – Seite 44]).
    - hash: Der gewählte Anker kann durch die Eigenschaft hash gelesen bzw. gesetzt werden (mit führendem „#“).
    - reload(): Mit der Methode reload() wird die URL neu angefordert und somit eine Aktualisierung der Seite erreicht. Dies ist im Verhältnis zu den vorhergehenden Eigenschaften eher harmlos. Ein Angriff könnte die Seite aber auch zum Dauer-Aktualisieren bringen.
    - replace(): Durch die Methode replace() wird das Aufrufen einer neuen URL veranlasst. Der Unterschied zum Setzen von location.href besteht darin, dass auch der aktuelle Eintrag in der Browser-Historie überschrieben wird.
- navigator.plugins: Durch das plugins Objekt, ein Unterobjekt des navigator (Browser) Objekts, kann man ermitteln, welche Plugins der Anwender im Sinne der Netscape-Plugin-Technik installiert hat. Dies kann einem Angreifer Informationen liefern, welche Schwachstellen der Browser des Opfers aufweisen könnte und welche zusätzlichen Sicherheitsplugins installiert sind.
- window: Das window Objekt repräsentiert das Fenster, in dem eine Seite dargestellt wird. Es selbst ist eher unkritisch aber eine seiner Methoden ist zu beachten.
    - window.open(): Die Methode open() des window Objekts öffnet ein neues Fenster. Der Aufruf wird heute zumeist durch Popupblocker verhindert.

- Alle Eigenschaften, die eine URL repräsentieren oder Teile einer URL sind, wie z. B. „src“ im IMG-Tag.

Da sich die JavaScript-Implementierungen der einzelnen Browser unterscheiden, kann es noch weitere Objekte, Methoden oder Eigenschaften geben, die für einen Angreifer interessant sein können. Außerdem kann sich auch der JavaScript-Umfang von einer Browser-Generation zur nächsten ändern. Deshalb muss für jeden Browser eine eigene Analyse durchgeführt werden, wo noch Gefahren bestehen.

Skripte können überall im Dokument zwischen den Script-Tags „<SCRIPT>“ und „</SCRIPT>“ stehen. Dabei kann im öffnenden Tag mit dem Attribut „src“ eine Quelle angegeben werden, von der Script-Code geladen werden kann. Des Weiteren kann JavaScript bei jedem Event-Handler verwendet werden. Außerdem kann JavaScript überall mit „javascript: …;“ angegeben werden, wo URLs aufgerufen werden können.

2.1.7.2 Beachtenswerte HTML-Elemente

Nicht nur JavaScript kann gefährlich sein. Selbst reines HTML kann gefährliche Elemente enthalten. Nachfolgend werden einige gefährliche Elemente aufgeführt und erklärt, warum diese gefährlich sein können.

- <META http-equiv=“refresh“ content=“0; URL=http://boese-seite.example.com/”>: Mit einem META-Tag, bei dem das Attribut „http-equiv“ auf „refresh“ gesetzt ist, wird eine automatische Weiterleitung erreicht. Im Attribut „content“ wird dabei festgelegt, wie viele Sekunden die wirklich aufgerufene Seite angezeigt wird und welche URL danach aufgerufen wird.
- Alle Elemente, die URLs enthalten können, können Informationen an andere Seiten weitergeben. Hierbei gibt es offensichtliche und weniger offensichtliche Attribute. Nachfolgend einige Beispiele:
    - Hyperlinks: <A href="…">
    - Ausschnitte für verweissensitive Grafiken: <AREA href="…">
    - Frames: <FRAME src="…">
    - IFrames: <IFRAME src="…">
    - Multimediaobjekte, wie Flash: <OBJECT codebase="…" classid="…" />
    - Plugin-Referenz: <EMBEDED src="…" />
    - Java-Applets: <APPLET codebase="…" code="…" archive="…" />
    - Formulare: <FORM action="…">
    - Bilder: <IMG type="image" src="…" lowsrc="…" />
    - Grafische Buttons: <INPUT src="…">
    - Skripte: <SCRIPT src="…"> (Selbst wenn JavaScript deaktiviert ist, da Informationen über die URL-Parameter im src-Attribut übergeben werden können!)
    - Logische Beziehungen zu anderen Dateien: <LINK href="…">
    - Hintergründe bei Tabellen: <TABLE background="…">
    - Hintergründe allgemein: Die CSS-Attribute "background" und "background-image"

Direkte Angriffe mit HTML-Elementen sind nur bei Attacken möglich, die über den Server laufen, also bei reflektiertem und persistentem XSS. Bei DOM-basiertem XSS muss der Client zunächst noch die gefälschten Parameter der URL verarbeiten. Dies ist ohne aktive Inhalte nicht möglich. Deshalb kann bei DOM-basierten XSS auch direkt schädlicher Skript-Code ausgeführt werden.

2.1.8 Verschleierung

Da einige Angriffe sehr leicht erkennbar sind, kann der Angreifer verschiedene Verschleierungstaktiken anwenden. Wenn man eine URL wie „http://goodsite.example.com?q=<SCRIPT>…</SCRIPT>“ sieht (URLs werden i. d. R. in der Statusleiste angezeigt, wenn man über einen Link fährt), wird der eine oder andere vielleicht stutzig. Deshalb nutzen Angreifer teilweise sehr lange URLs, die dann nicht mehr vollständig angezeigt und auch nicht anderweitig überprüft werden. Außerdem können reflektiertes und persistentes XSS dazu führen, dass die URL in einem HTML-Element verwendet werden, das diese selbstständig aufruft, wie beim IMG-Tag.

2.1.8.1 Automatische Filter

Manchmal werden einfach automatische Filter verwendet. Wird z. B. nur nach „<SCRIPT>“ gefiltert, kann solch ein Filter u. U. mit „<<SCRIPT >“ umgangen werden, wenn alles zwischen „<“ und „>“, also „<SCRIPT“ mit „SCRIPT“ verglichen wird.

Löscht ein Filter jedes Vorkommen von „SCRIPT“ (nicht-rekursiver Filter), kann „<SCRSCRIPTIPT>“ zum Erfolg führen.

Auch das Verwenden von anderen Zeichencodierungen, wie UTF 8, kann teilweise Filter austricksen.

Werden URL-Parameter in Event-Handler eingebunden, ist kein SCRIPT -Tag notwendig.

2.1.8.2 Eingeschränkte Zeichenlänge

Eine eingeschränkte Zeichenlänge kann zumeist mit „<SCRIPT src= … >“ umgangen werden oder dadurch, dass mehrere Attribute, z. B. bei Formularen, für einen XSS-Angriff genutzt werden. Hierbei werden zumeist „/*“ und „*/“ als Metazeichen für den Kommentarbereich innerhalb von Script-Bereichen eingesetzt.

2.2 Cross-Site Request Forgery (CSRF)

Zur Bearbeitung dieses Angriffs orientierte ich mich an den Quellen (Computerwoche, 2007; Huseby, 2004; OWASP Foundation, 2008). Eine Cross-Site Request Forgery (gefälschte Anfrage) ist eine Attacke, bei der sich ein Angreifer eines Opfers bedient, um eine Webanwendung eine gewünschte Aktion ausführen zu lassen. Das Opfer muss hierzu normalerweise bei der Webanwendung angemeldet sein. Es gibt zwei grundsätzlich unterschiedliche Mittel, den Browser des Benutzers zum Aufrufen einer gewünschten URL zu bringen. Zum einen kann dies mit dem Wissen des Anwenders erfolgen, d. h. der Benutzer führt dies selbst durch, indem er auf einen Link klickt oder ein Formular abschickt. Dies kann zumeist, wie bereits bei XSS erwähnt, durch Social Engineering erreicht werden. Zum anderen kann der Aufruf der URL unbemerkt vom Anwender erfolgen. Hierzu muss der Angreifer nur einen Link, der automatisch von Browsern aufgerufen wird, auf eine Webseite schmuggeln, die vom Opfer aufgerufen wird. Dies hört sich komplizierter an, als es ist. Der Angreifer kann z. B. einen IMG-Tag mit der vom Opfer aufzurufenden URL im src-Attribut in einer Webseite einbinden, die das potentielle Opfer aufruft. Anstatt eines Links kann er auch JavaScript auf die Seite einbauen, die das Opfer aufruft. Allerdings ist das Einbinden eines Bildes meist einfacher. Eine weitere Möglichkeit ist das Versenden einer HTML-Mail mit einem manipulierten HTML-Element, wie einem IMG-Tag, der kein Bild sondern eine Aktion aufruft.

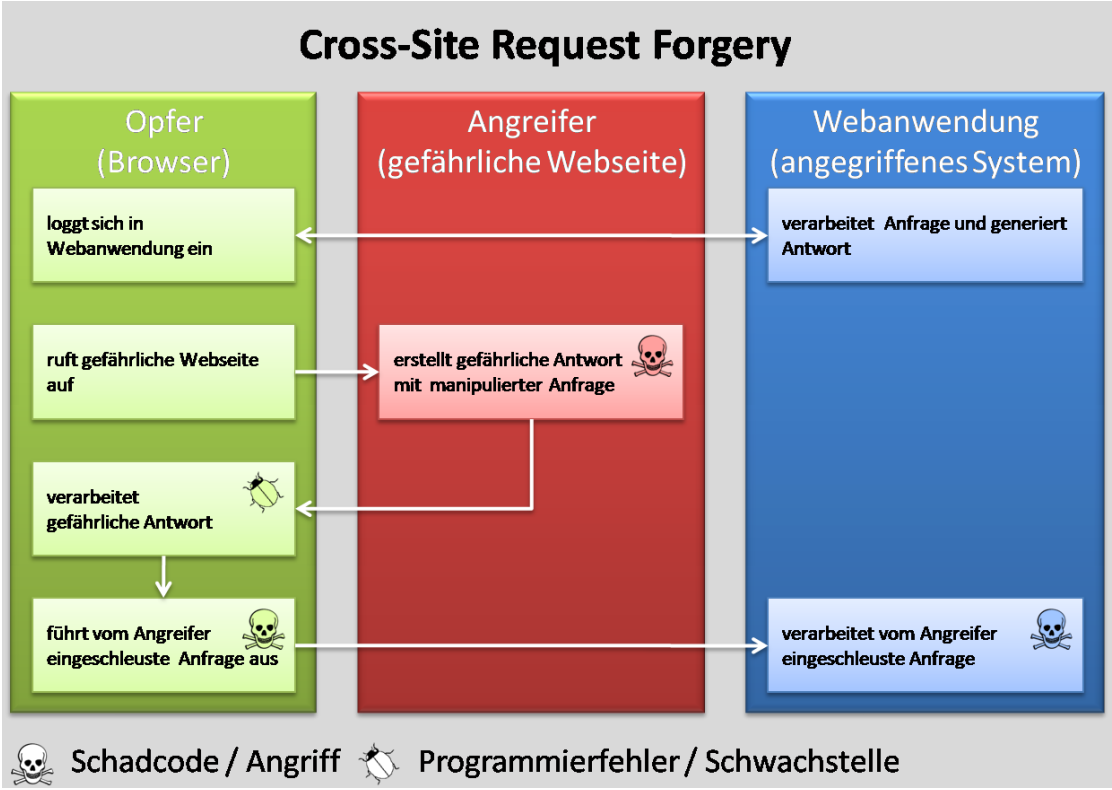

Abbildung 2-4: Cross-Site Request Forgery

2.2.1 Ziele von CSRF

Das Ziel des Angreifers ist immer die Manipulation einer Webanwendung. Allerdings kann die Absicht dahinter von verschiedenster Art sein. Hat das Opfer z. B. die Privilegien, einen neuen Benutzer für die Anwendung anzulegen, kann der Hacker dies zu seinen Gunsten ausnutzen um Zugriff auf die Webanwendung zu erhalten. Bei Online-Auktionshäusern kann er das Opfer dazu bringen, unwissentlich für einen Artikel zu bieten. Bei Communities kann ein gefälschter Link dazu führen, dass das Opfer unbewusst seine Profilseite verändert und beispielsweise ein Link darauf erscheint, der gar nicht wissentlich vom Opfer eingestellt wurde. Freunde des Opfers klicken somit möglicherweise auf den Link und werden selbst Opfer eines Angriffs oder gelangen auf dubiose Internetseiten. Eine der bekanntesten Möglichkeiten ist das Abschalten der Firewall oder Ändern der Konfiguration des Routers des Opfers. Dadurch wird das lokale Netzwerk leichter angreifbar und der Hacker kann Sicherheitslücken ausnutzen, um den Rechner des Opfers zu kompromittieren. In Foren, Blogs oder Gästebüchern kann ein Angreifer so Einträge im Namen des Opfers hinterlassen. Dies kann einerseits zur Rufschädigung und Ähnlichem genutzt werden oder andererseits zum Missbrauchen des Vertrauens gegenüber anderen Anwendern in Bezug auf die getätigten Einträge, die beispielsweise gefährliche Links enthalten können. Ist ein Opfer gerade in einem Online-Mailer eingeloggt, kann der Angreifer auch Mails über dessen Konto schicken. In Online-Banking-Anwendungen kann unter anderem eine Überweisung getätigt werden. Die möglichen Aktionen hängen davon ab, welche Webanwendung angegriffen wird und welche Sicherheitsmechanismen diese verwendet.

2.2.2 Abgrenzung zu XSS

Der Angriff dient lediglich zum Ausführen gewünschter Aktionen in Webanwendungen und greift diese nicht selbst an. Auch das Ausspionieren von Daten ist somit nicht möglich, da die Anfrage vom

Browser des Clients an den Server geschickt wird und von diesem eine Antwort wieder zurück zum Client. Außerdem wird auch kein direkter Angriff auf das Cookie des Anwenders unternommen.

Wegen der Namensähnlichkeit wird Cross-Site Request Forgery meist fälschlicherweise Cross-Site-Scripting untergeordnet. Außerdem werden die Angriffe meist mit Hilfe derselben HTML-Elemente, JavaScript-Objekte, -Eigenschaften oder -Methoden durchgeführt, was zu Verwechslungen führen kann. Allerdings unterscheiden sich beide Angriffe in wesentlichen Punkten: Erstens reicht bei CSRF meist der Aufruf einer URL aus, die keine manipulativen Parameter beinhaltet. So ist solch ein Aufruf für den Server nicht von einem „normalen“ Aufruf des Clients zu unterscheiden, da kein ausführbarer Code in die Webanwendung eingeschleust werden muss (dies wäre eine Interpreter-Injection). Somit lässt sich CSRF auch ausführen, wenn eine Anwendung gegen XSS resistent ist! Zweitens ist bei XSS das Ausspähen von Daten und ggf. die Übernahme der Session des Anwenders das Hauptziel. Hierbei müssen Daten irgendwie an den Angreifer zurückgesendet werden (meist durch Aufrufen einer URL, einer Webseite, die vom Hacker kontrolliert wird mit den Daten als URL-Parameter). Bei CSRF ist das Hauptziel aber das Ausführen von Aktionen in Webanwendungen, bei denen das Opfer angemeldet ist. In diesem Fall ist es nicht nötig, dass der Hacker selbst Daten übermittelt bekommt.

Besonders gefährlich für Anwender ist aber die Kombination aus XSS und CSRF. Hierbei werden die gefälschten Links mit Hilfe von XSS direkt in die zu manipulierende Webanwendung geschleust. Da die Links für den Client nicht von „normalen“ Links der Anwendung zu unterscheiden sind, kann er sie nicht als unsicher erkennen und wird ihnen vertrauen.

2.2.3 Schwachstellen

Bei Cross-Site Request Forgery werden hauptsächlich HTML-Elemente verwendet, die in einem Attribut eine URL akzeptieren, siehe auch [Beachtenswerte HTML-Elemente – Seite 20]. Außerdem ist auch die Verwendung von JavaScript nicht ausgeschlossen. Hierbei sind vor allem das location-Objekt mit seinen Eigenschaften und Methoden sowie die Methode window.open zu beachten, siehe auch [Beachtenswerte JavaScript-Funktionen – Seite 18]). Für den Benutzer besonders gefährlich ist, wie bereits erwähnt, wenn XSS benutzt wird, um manipulierte Links in Webanwendungen einzuschleusen, die einen CSRF-Angriff umsetzen.

2.2.4 Beispiele

- Ausschalten der Firewall im Linksys-Router WRT54GL (Bachfeld, 2008):
  https://192.168.1.1/apply.cgi?submit_button=Firewall&change_action=&action=Apply&block_wan=1&block_loopback=0&multicast_pass=0&ident_pass=0&block_cookie=0&block_java=0&block_proxy=0&block_activex=0&filter=off&_block_wan=1&_block_multicast=0&_ident_pass=1
- Bei einer unsicheren Bank kann mit Hilfe von CSRF eine Buchung erfolgen. Ein ähnlicher Angriff war bei der amerikanischen Direktbank ING möglich (Bachfeld, 2008):
  <IMG src=“https://insecure-bank.example.com/transfer?betrag=5000&konto=123456“>
- Ausloggen eines Benutzers aus einer Community.
  <INPUT src=“http://community.example.bank/perform?action=logout” />

2.2.5 Maßnahmen, die CSFR nicht abwehren

POST statt GET verwenden:

Die meisten Entwickler glauben, wenn man nur POST-Anfragen verarbeitet, ist CSRF ausgeschlossen. Dies ist aber nicht richtig, da man auch Formulare fälschen bzw. automatisiert abschicken kann. Zwar

ist das Einschleusen eines URL-Aufrufs wesentlich einfacher, allerdings spricht nichts dagegen, dass ein solcher Aufruf dann auf eine Seite führt, die ein automatisiertes Formular absendet.

Trotzdem ist es meist schwerer, Formulare zu fälschen als URL-Parameter. Hier eine Empfehlung, wann man GET und wann man POST verwenden sollte:

- Verwenden Sie GET bei Aktionen, die eine Anfrage, einen Lese-Zugriff oder ein Nachschlagen darstellen. GETs werden von Suchmaschinen und Spidern verfolgt und in der Browser-Historie gespeichert.
- Benutzen Sie POST, wenn Aktionen eine haftbare Handlung, einen Auftrag oder eine Änderung repräsentieren.

2.2.6 Abwehrmaßnahmen

2.2.6.1 serverseitig

- Einschränken der akzeptierten Methode
- Verwenden von Tickets, die nur eine Anfrage lang gültig sind

2.2.6.2 clientseitig

Ein Browser könnte URLs überprüfen und, z. B. überall wo ein Bild erwartet wird, einschränken. So könnte eine URL, die auf „.jpg“, „.png“, „.gif“ oder eine andere sinnvolle Dateierweiterung endet, erwartet werden. Außerdem sollte danach kein „?“ folgen. Falls diese Regel zu strikt ist, z. B. wenn ein Webserver eine Variable mit „size=small“ erlaubt, könnte dies explizit angepasst werden. Leider erfordert ein solcher Filter und dessen Pflege einigen Aufwand.

Eine weitere Möglichkeit ist eine bessere Cookie-Verwaltung im Browser zu implementieren. So könnte der Browser die Cookies für jedes offene Tab oder Fenster separat verwalten. Außerdem könnte er das Cookie wieder löschen, sobald im Tab eine andere Domäne aufgerufen wird. Da der Anwender zumeist über das Cookie authentifiziert wird, schlägt so der Angriff von einer externen Seite fehl.

2.3 GIFAR (JAR-Archiv im GIF)

GIFAR ist ein neuartiger, noch zum Großteil unbeachteter Angriff. Ein Angreifer kombiniert dabei ein Bild im GIF-Format mit einem JAR-Archiv, das ein von ihm entwickeltes Applet enthält. Dieses schiebt er einem Server unter, auf dessen Nutzer er es abgesehen hat, indem er die kombinierte Datei hochlädt (Bachfeld, 2008). In vielen Webanwendungen ist das Hochladen von Bildern möglich. Ruft nun ein Benutzer, besser gesagt dessen Browser, das Bild in normaler Weise auf, also indem es z. B. in einem IMG-Tag als Quelle angegeben ist, erscheint es für ihn als normales Bild. Soweit ist dies nichts Interessantes. Wenn der Angreifer es allerdings ermöglichen kann, dass das in der kombinierten Datei enthaltene Applet aufgerufen wird, ist dies schon ein größerer Erfolg für den Unruhestifter. Genau dies kann er erreichen, indem er es auf einer **beliebigen** Seite als Applet mit dem entsprechenden Tag einbindet. Um das Opfer dazu zu bringen, diese Seite aufzurufen, kann der Angreifer z. B. Social Engineering, XSS oder CSRF verwenden. Wenn im Browser des Benutzers Java aktiviert ist, wird nun nicht das Bild angezeigt, sondern das Applet aufgerufen!

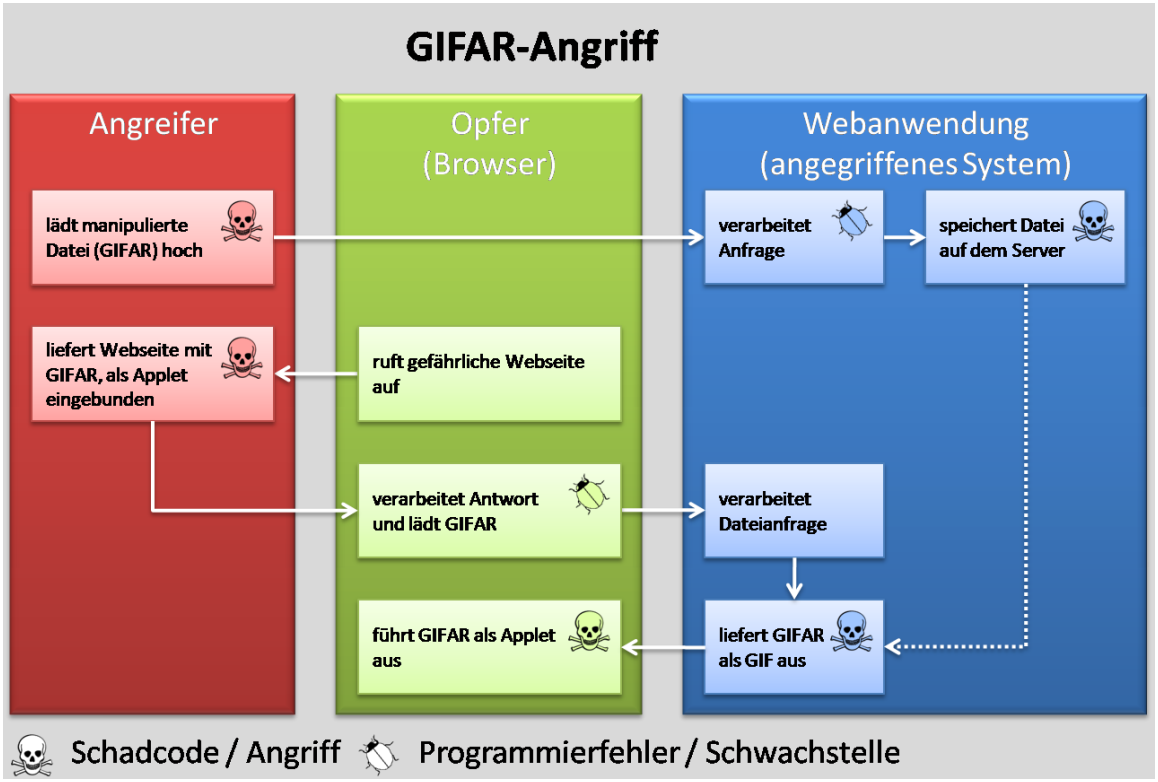

Abbildung 2-5: GIFAR-Attacke

Der Hacker kann somit bestimmen, was der Client ausführen soll, jedoch sind Applets eingeschränkt. Allerdings kann ein Applet mit der Domäne, von der es stammt kommunizieren. Netterweise werden dabei auch die Cookies mitgesandt, die der Browser für die entsprechende Domäne gespeichert hat. So sind also CSRF-Angriffe möglich. Außerdem kann der Hacker auch Daten dieser Domäne, auf die der Benutzer Zugriff hat, über das Applet ausspionieren. Ein Hacker kann das Applet sogar als Proxy verwenden, um auf die Webanwendung Zugriff zu erhalten, so lange das Opfer in der betreffenden Webanwendung eingeloggt ist und das Applet aufgerufen hat.

Falls sich der infizierte Client innerhalb eines für die Firewall des Servers vertrauenswürdigen Netzes befindet, kann der Angreifer diese sogar durchbrechen. Näheres hierzu in [Überwinden der Firewall – Seite 27]. Weil Services im Backend, wie Datenbanken, teilweise schwach gesichert sind, bietet dies dem Angreifer neue Angriffsmöglichkeiten auf diese zum Teil schwach gesicherten Dienste.

Kann dieser Angriff zusammen mit persistentem XSS kombiniert werden, hat der Nutzer nicht einmal die Möglichkeit, solch einen Angriff zu erkennen. Wenn sich durch die XSS-Schwachstelle das Applet z. B. ins eigene Profil einbinden lässt, kann es sich durch Ausnutzen von CSRF selbst in der Webanwendung ausbreiten, wie dies Computer-Würmer tun. Je nachdem, wie geschickt der Angreifer dies implementiert, kann sich das Applet sogar über Webanwendungen hinweg ausbreiten, wenn diese selbst Schwachstellen aufweisen, was geradezu katastrophale Auswirkungen hervorrufen kann.

2.3.1 Ziele

Die Ziele des Angreifers können bei diesem Angriff so vielfältig sein, wie fast bei keinem anderen. Die beliebtesten Angriffe sind die auf Nutzer. So kann durch GIFAR-Attacken das Cookie geklaut werden. Dies führt dazu, dass der Angreifer die Session des Opfers mitverwenden kann. Doch selbst wenn dies durch spezielle Schutzmaßnahmen nicht möglich wäre, kann das Applet selbst http-Requests im

Namen des Opfers absenden und so CSRF-Angriffe ausführen. Diese können über einfache Manipulationen bis hin zur Übernahme des Benutzerkontos durch den Hacker reichen. Außerdem kann der Hacker durch Implementierung des Applets als Proxy aktiv Daten, auf die das Opfer Zugriff hat, ausspionieren oder gar verändern. Dies ist beispielsweise über ein sogenanntes Zone-Bridging möglich. Hierbei verbindet das Applet die Zone der Domäne, von der es stammt und zu der es also Verbindungen aufbauen kann, mit der Zone der Webseite, in der es eingebunden wurde und mit der es z. B. über JavaScript kommunizieren kann. Auch eine Kommunikation über http kann ermöglicht werden, was aus Sicht der Angriffsabwehr wesentlich gefährlicher ist. JavaScript kann man blockieren, http-Verbindungen sind allerdings die Grundlage für die Kommunikation des Browsers mit dem Webserver und können deshalb nicht unterbunden werden. Die genannten Techniken lassen sich auch sehr gut verwenden, um das Opfer in Schwierigkeiten zu bringen und so z. B. indirekt dessen Benutzerkonto sperren zu lassen.

Allerdings betreffen all diese Angriffe meist nur einzelne Benutzer. Schafft es der Angreifer aber, dass das Applet von einer Zone, der der Server und dessen Firewalls vertrauen, aufgerufen wird, so kann er durch dieses Loch das System angreifen. Direkte Angriffe auf das Backend sind bisher kaum möglich, da dieses normal durch spezielle Firewalls gesichert und so vor externen Zugriffen geschützt ist. Da der Schutz (z. B. Verwenden starker Passwörter) von bestimmten Diensten, wie Datenbankserver usw., meist nicht so ernst genommen wird wie der von Webservern, lassen sich hier unter Umständen leichter Sicherheitslücken entdecken und ausnutzen.

Viele der genannten Angriffe sind auch beispielsweise über JavaScript möglich. Da dies aber meist aus Profilen ausgefiltert oder gar ganz blockiert wird, ist es zumeist schwierig, diese umzusetzen. Hochgeladene Dokumente werden allerdings oft weniger strikt kontrolliert. Deswegen kann die GIFAR-Attacke auch dazu verwendet werden Angriffe auszuführen, die durch JavaScript-Filter verhindert werden. Hierzu muss der Hacker den Server davon überzeugen, dass er ihm eine reguläre Datei übergibt. Deshalb ist eine Kombination einer regulären Datei mit einem JAR-Archiv nötig.

2.3.2 Schwachstellen

Alle Anwendungen, die das Hochladen von Dateien in einem Dateiformat erlauben, das auf ZIP basiert, sind anfällig für GIFAR-Attacken. Zwar lässt der Name vermuten, dass die Attacke speziell ein GIF-Bild und eine JAR-Datei kombinieren muss, allerdings ist dies nicht der Fall. Der erste veröffentliche Beispielangriff wurde aber mit einer Kombination aus GIF-Bild und JAR-Archiv bekannt gemacht, daher der Name. Andere auf ZIP basierende Dateiformate sind z. B. das Open Document Format (.od*), das Office Open XML Format (.docx, .xlsx, .pptx), JPEG (.jpg, .jpeg) usw.

Einige Webanwendungen wandeln zwar die Dateien um, indem sie die Bilder z. B. auf fest definierte Größen skalieren, speichern jedoch die Originaldatei trotzdem, um das Bild in Originalqualität anzeigen zu können.

Für den Angriff auf das Backend muss das System zusätzlich schwache Firewall-Regeln aufweisen bzw. offen sein.

Beispiele für verwundbare Webapplikationen sind Communities, Onlineauktionshäuser, Bildergalerien und viele andere Webseiten, bei denen die Benutzer den Inhalt erzeugen und meist auch eine eigene Profilseite haben.

2.3.3 Beispiele

2.3.3.1 Applet zum Manipulieren des Benutzerprofils

Nehmen wir an, der Angreifer ist Mitglied bei einer Online-Community, in der jeder Anwender eigene Fotoalben pflegen kann. Er sucht sich ein Foto namens „tolles_Bild.jpg“ aus. Außerdem packt er sein Applet in das JAR-Archiv „boeses_Applet.jar“. Das Applet löscht alle Fotoalben des Benutzers, der es aufruft. Danach kopiert er beide Dateien in eine Datei mit dem Namen „mein_Bild.jpg“ zusammen. Dabei muss das Archiv „boeses_Applet.jar“ hinter das Bild „tolles_Bild.jpg“ angehängt werden. Nun lädt er in eines seiner Fotoalben die präparierte Datei „mein_Bild.jpg“ hoch (Petkov, 2007). Danach muss er nur noch einen Nutzer dazu bringen eine präparierte Seite aufzurufen, die dieses Applet korrekt einbindet. Ist ein Benutzer in der Webanwendung eingeloggt und ruft dann diese Seite auf, so löscht das Applet in seinem Namen seine gesamten Fotoalben. Der Anwender merkt dies nicht direkt und kann das Problem wohl auch nicht zuordnen. Wenn er nun keine Backups hat (welcher private Nutzer hat diese schon?) und sich darauf verlassen hat, dass die Bilder in der Anwendung sicher sind, diese also auch nicht selbst auf dem privaten PC gespeichert hat, sind seine Fotos verloren.

2.3.3.2 Überwinden der Firewall

Um die Backendfirewall zu überlisten, muss der Angreifer die Möglichkeit haben, eine Verbindung zu einem Server aus einem für die Firewall vertrauenswürdigen Netzwerk aufzubauen. Dies ist i. d. R. für ihn nur indirekt möglich, da er sich normalerweise nicht in einem solchen befindet. Jedoch kommen auch immer mehr Angriffe von unzufriedenen Mitarbeitern, die sich im Intranet und somit im teilweise vertrauenswürdigen Netz befinden können. Je nach Größe und Sicherheitskonzept der Firma kann der Angriff also auch direkt aus einem vertrauenswürdigen Netz erfolgen.

Nehmen wir an, der Hacker lädt die gefälschte Datei aus einem nicht vertrauenswürdigen Netz auf den Server www.example.com hoch. Der Server www.example.com befindet sich hinter einer Firewall, die nur http-Verbindungen von außerhalb erlaubt. Neben dem Webserver befindet sich auch der Datenbankserver auf dem Rechner. Die Firewall-Regeln sehen für Verbindungen von innerhalb des Mitarbeiternetzes keine Einschränkungen vor. Schafft es nun der Angreifer, dass die Seite, die sein eingeschleustes Applet aufruft, von einem Mitarbeiter angesurft wird, kann er die Datenbank und andere Dienste angreifen, die auf dem Webserver laufen, da das Applet Socket-Verbindungen zum Server aufbauen kann. Dadurch, dass diese Verbindungen in unserem Beispiel von der Firewall nicht unterbunden werden, weil sie aus dem normalerweise vertrauenswürdigen Mitarbeiternetz kommen, öffnet dies neue Angriffsmöglichkeiten für den Hacker. Viele lokale Dienste sind meist nicht ausreichend gesichert, so dass eine Attacke hier leichter zum Erfolg führen kann. Die Folgen können hierbei sehr weitreichend sein und sogar zur Übernahme des Servers führen.

2.3.4 Abwehrmaßnahmen

Ein Webserver sollte immer eine Überprüfung der hochgeladenen Dateien vornehmen. So sollten z. B. die Dateiheader begutachtet werden. Des Weiteren sollte man Bilder in verschiedene Formate und Größen konvertieren. Bei anderen Dateien, wie Dokumenten im Office Open XML Format, müssen diese geöffnet und wieder gespeichert werden. In beiden Fällen verschwindet der „Anhang“. Wichtig ist auch, dass dann die Originaldatei verworfen wird.

Ist ein Cookie als httpOnly gekennzeichnet, so wird es bei Anfragen des Applets nicht mit zum Server gesandt.

Auch dem Browser stehen Abwehrmaßnahmen zur Verfügung. Dieser sollte vor allem den MIME-Typ überprüfen. Da der Server nicht weiß, wie der Client die Datei einbindet, trifft ihn eine geringere Schuld als den Browser. Er hält die gesendete Datei für ein Bild bzw. ein anderes Dokument und gibt dies i. d. R. auch im Content-Type-Header an. Überprüft der Browser diesen und erhält einen MIME-Type für einen APPLET-Tag, der nicht für die Java Virtual Machine gedacht ist, wie z. B. „image/gif“, so sollte er die Antwort verwerfen und den Inhalt blockieren.

Außerdem sollte der Anwender Java nur auf den Seiten aktivieren, auf denen er es unbedingt benötigt. Wenn Java für die Seite des Angreifers deaktiviert ist, schlägt die Attacke fehl.

Eine weitere Möglichkeit wäre, dass der Browser standardmäßig nur Applets lädt, die vom gleichen Server kommen und andere nur, wenn der Benutzer eine Ausnahme hinzufügt. So wird verhindert, dass die gefährliche Seite das eingeschleuste Applet vom manipulierten Server laden kann.

2.4 Clickjacking

Clickjacking oder auch UI Redressing ist keine neue Angriffstechnik. Sie wurde bereits 2002 erwähnt (O'Callahan, 2002), allerdings wurde das Potential von Clickjacking stark unterschätzt. Letztes Jahr wurde das Thema vom OWASP wieder neu aufgegriffen und in den Medien diskutiert(Rütten, 2008; Bachfeld, 2008). Bei dieser Angriffstechnik ist das Ziel, den Benutzer Klicks ausführen zu lassen, ohne dass ihm bewusst wird auf was er gerade wirklich geklickt hat. Dies kann z. B. über ein Browsergame erreicht werden. Der Anwender denkt, er bedient nur das Spiel. In Wirklichkeit überlagern aber andere Elemente dieses und werden so angeklickt. Hiermit kann der Angreifer alles erreichen, wozu ein Klick vom Benutzer nötig ist.

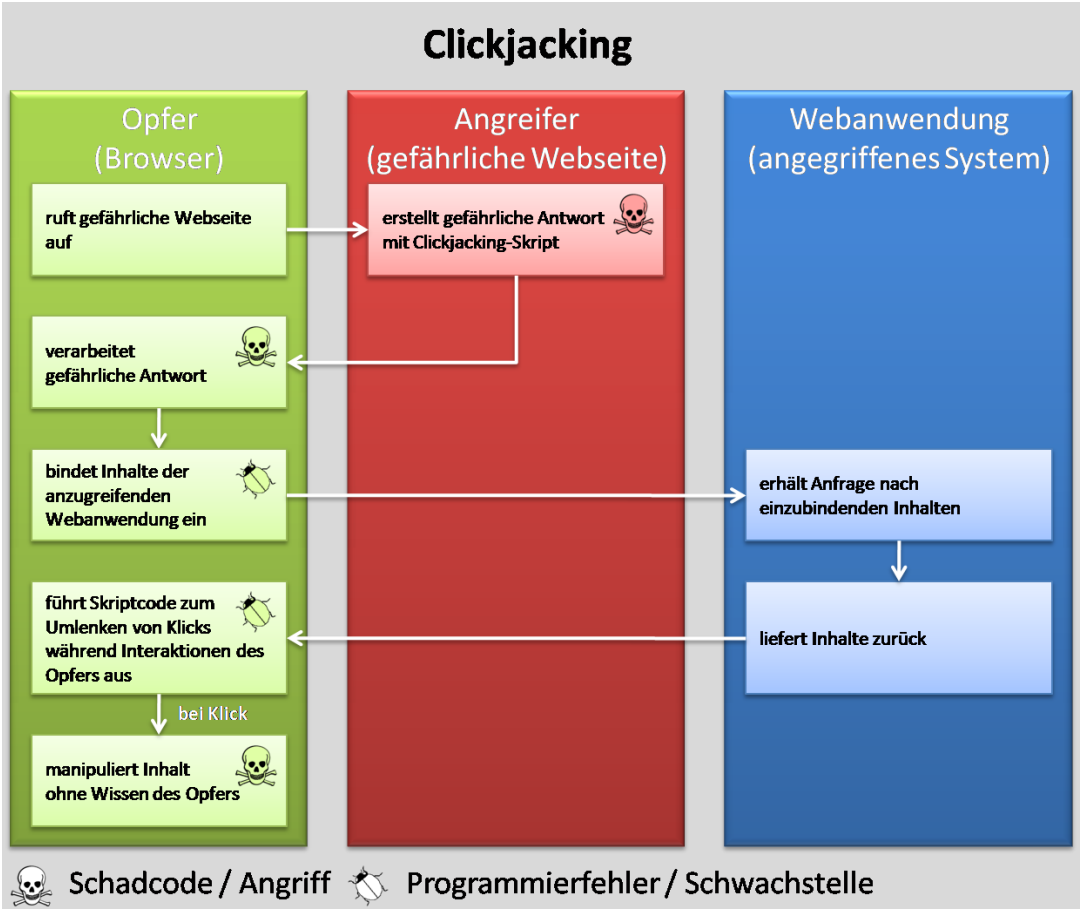

Abbildung 2-6: Clickjacking

2.4.1 Ziele

Durch Clickjacking kann ein Angreifer z. B. erreichen, dass der Anwender XSS-verseuchte Links anklickt, die dann einen weiteren Angriff implementieren. Genauso kann ein entführter Klick zum Ausführen einer CSRF-Attacke führen. Dies ist wohl wenig spektakulär, da es für die genannten Angriffe auch geeignete Abwehrmaßnahmen gibt.

Schützt eine Anwendung Anfragen z. B. mit einem Ticket, das zufällig gewählt wird und nur in der Session auf dem Server, sowie in den vom Server ausgelieferten Seiten enthalten ist, so ist CSRF normalerweise nicht durchführbar. Der Server wird eine Anfrage, die kein gültiges Ticket beinhaltet, einfach verwerfen. Ein solches Ticket aber zu erraten ist i. d. R. kaum oder gar nicht möglich. Verwendet ein Angreifer nun allerdings Clickjacking, wird eine reguläre Seite von der Webanwendung angefordert, die auch ein gültiges Ticket enthält. Allerdings wird diese Seite geschickt versteckt. Wenn nun das Opfer auf das vom Angreifer beabsichtigte Ziel klickt, sendet der Browser einen gültigen Request an die Anwendung. Für diese ist nicht erkennbar, dass es sich um einen CSRF-Angriff handelt.

Des Weiteren kann durch Clickjacking erreicht werden, dass sich ein Benutzer in eine Anwendung einloggt. Wenn der Benutzer einen Passwortmanager benutzt, ist es sogar möglich, dass hierfür ein einziger Klick ausreicht, da die Anmeldedaten bereits vom Passwortmanager eingefügt werden. Aber nicht nur das Einloggen ist möglich, sondern dient meist nur als Mittel zum Zweck. Ist ein Anwender erst einmal eingeloggt, kann der Angreifer beispielsweise Einstellungen verstellen, das Benutzerkonto löschen, im Namen des Benutzers einkaufen, anrufen, für einen Artikel bieten usw. Was alles möglich ist, hängt von der jeweiligen Anwendung selbst ab. In Webanwendungen wie Outlook Web Access kann der Hacker z. B. Mails des Benutzers weiterleiten. Dies kann vor allem von Interesse sein, falls der Angreifer eine „Passwort zusenden“-Funktion aufruft und danach die Mail an sich weiterleitet. Da die Mailadresse aber bei verschiedenen Webanwendungen ohne Authentifizierung geändert werden kann, kann der Angreifer auch über eine Änderung dort das Passwort zugesendet bekommen. Gewöhnliche Sicherheitsabfragen können dabei leicht durch weiteres Clickjacking bestätigt werden. Auch die meisten Router und Firewalls mit Webfrontend sind von solchen Angriffen betroffen.

2.4.2 Schwachstellen

Alle Seiten, die mit oder ohne JavaScript in einem Frame darstellbar sind, sind anfällig für Clickjacking. Konkret sind vor allem Links und Absende-Buttons dieser Seiten die Ziele, auf die es ein Angreifer normalerweise abgesehen hat. Bekannt gewordene Angriffe für dieses Szenario sind z. B. die Google-Desktop-MITM-Attacke, sowie ein Angriff zum automatischen Hinzufügen von Gatgets und zum Klickbetrug bei Google Gadgets. Angaben hierzu und zu den nachfolgenden Schwachstellen sind unter (Hansen, 2008) zu finden. JavaScript ist hierbei nicht von Nöten, da sich auch mit Cascading Stylesheets IFrames unsichtbar machen lassen. Deaktiviertes JavaScript verschlimmert sogar die Lage des Benutzers, da Frame-Bursting nicht mehr funktioniert. Bei Frame-Bursting ersetzt ein JavaScript die URL des Fensters („_top“) mit der seines Frames, wenn sich beide unterscheiden. Somit wird vermieden, dass diese Seite innerhalb eines Frames geladen wird (wenn JavaScript aktiviert ist).

Beispielcode für Framebursting:

```
<SCRIPT>if (top != self) top.location = location</SCRIPT>
```

Code-Beispiel 2-15: Famebursting: JavaScript-Beispiel

Auch ActiveX-Komponenten sind für Clickjacking anfällig. Da für diese kein domänenübergreifender Zugriff notwendig ist, sind sogar IFrames und Frames unnötig. Der Angreifer kann Flash beispielsweise über einen OBJECT-Tag einbinden und darüber einen DIV legen (Hansen, et al., 2008). Ein Beispiel hierzu ist ein Hack, bei dem die Mikrofon- und Kameraeinstellungen des Opfers in Flash geändert werden (Aharonovsky, 2008). Dieser Exploit ist bereits nicht mehr funktionsfähig. Es sind allerdings auch die Sicherheitseinstellungen von Flash von diesem Problem betroffen.

Bei aktiviertem JavaScript, kann der Status von Klicks auf Objekte einer anderen Domäne überwacht werden. Dies erlaubt komplexere Angriffe mit einer Folge aus mehreren Klicks.

Frames können mit OBJECT-Tags realisiert werden, um IFRAME-Tag-Filter zu umgehen.

Mit Clickjacking können XSS-Schwachstellen ausgenutzt werden, die derzeit noch nicht ausgenutzt werden können, da die Sicherheitsbarrieren ohne Clickjacking nicht oder kaum überwunden werden können.

CSRF-Angriffe sind auch bei Webanwendungen möglich, die Anfragen durch Tickets schützen.

Benutzer, vor allem wenn sie Passwortmanager verwenden, können automatisch bei Webanwendungen angemeldet werden.

Manche Sicherheitsplugins, wie Flashblock für den Firefox, können durch Clickjacking ausgetrickst werden.

2.4.3 Abwehrmaßnahmen

Derzeit arbeiten viele Hersteller an geeigneten Schutzmaßnahmen. Webanwendungen sollten Frame-Bursting verwenden und ihre Tickets für Anfragen in JavaScript-Variablen speichern, so dass bei deaktiviertem JavaScript kein CSRF möglich ist.

Maßnahmen auf Seiten des Clients wären das Verbieten von IFrames und Plugins. Für den Browser Firefox gibt es das Sicherheits-Plugin NoScript, das Clickjacking weitestgehend erkennt und eine Warnung anzeigt.

Derzeit gibt es noch keine weiteren Maßregeln. Höchstens der Einsatz eines rein textbasierten Browsers, wie Lynx, wäre noch möglich, allerdings ist dies im Allgemeinen nicht gewünscht bzw. möglich. Auf vielen Seiten ist ein rein textbasierter Browser wenig hilfreich und außerdem für Otto-Normalverbraucher nicht komfortabel genug.

3 Abwehrmaßnahmen

Um Abwehrmaßnahmen implementieren zu können, ist es zuerst einmal nötig, sich darüber Gedanken zu machen, worin die Gefahren eigentlich bestehen, gegen die man sich verteidigen will. Diese Arbeit behandelt Abwehrmaßnahmen gegen Angriffe, die gegen Webanwendungen bzw. Benutzereingaben gerichtet sind. Bei Fehlern im Browser oder in Plugins desselben helfen die aufgeführten Maßnahmen nicht oder nur sehr eingeschränkt.

3.1 Einführung

3.1.1 Gefahren

Wie kann eine Attacke gegen Webanwendungen oder den Benutzer aussehen? Ein Angreifer kann versuchen, den Benutzer dazu zu bringen, gefälschte Formulare abzuschicken oder Links aufzurufen. Dies kann auch ohne Benutzerinteraktion geschehen, wofür der Browser so manipuliert werden muss, dass er es automatisch erledigt. Eine weitere Möglichkeit ist es, Webseiten zu verfälschen, um den Anwender zu verunsichern, in die Irre zu führen oder ihm eine falsche Ansicht zu vermitteln. Eines der beliebtesten Ziele ist allerdings das Klauen des Cookies und somit der Authentifizierungsdaten des Benutzers. Auch das Klauen anderer Daten, wie einer Kreditkartennummer, kann für Hacker interessant sein.

3.1.2 Objekte, die geschützt werden müssen

Ein Browser muss verhindern, dass ungewollte Anfragen abgesendet werden, vor allem wenn diese Benutzerdaten enthalten. Veränderte Ressourcen sind für den Browser i. d. R. nicht erkennbar, da er nicht im Voraus wissen kann, wie die angefragte Ressource auszusehen hat. Lediglich sehr grobe Informationen, wie erwarteter MIME-Typ, stehen ihm zur Verfügung. Außerdem ist der Browser für den besonderen Schutz der ihm anvertrauten benutzerspezifischen Daten, wie Cookies, gespeicherte Anmeldedaten, usw., verantwortlich und muss diese vor Missbrauch schützen. Der Browser ist unser Vermittler und darf uns nicht verraten.

3.1.3 Wie erreicht ein Angreifer sein Ziel?

Ein Angreifer kann eine Webseite anbieten, die Inhalte zum Stehlen benutzerspezifischer Daten, wie einem Session-Cookie, einbindet. Dazu muss der Angreifer zwei Ziele verwirklichen. Zum einen muss er die Sicherheitsbarrieren der Same-Orgin-Policy überwinden und zum anderen muss er den Benutzer dazu bewegen, seine Seite aufzurufen. Ersteres kann er durch einen Angriff wie XSS oder GIFAR realisieren, also durch Angriffe gegen die Webanwendung, letzteres durch Austricksen oder Überreden des Benutzers, beispielsweise mit Hilfe von Clickjacking oder Social Engineering.

Bei einem Angriff gegen die Webanwendung versucht der Angreifer entweder die Anwendung selbst durch einen ausführbaren Code oder interpretierte Zeichenketten zu kompromittieren, oder er versucht, ein Objekt in die Domäne der Webanwendung einzuschleusen, um den Browser des Opfers dazu zu bringen, dieses als vertrauenswürdig anzusehen und ihm so besondere Rechte einzuräumen, wie den Zugriff auf die Cookies der zugehörigen Domäne.

Bei Attacken gegen einen Benutzer geht es meist darum, diesen zu täuschen, indem man den dargestellten Inhalt verfälscht oder unsichtbar macht. Aber auch durch Ansprechen von Gefühlen wie Mitleid, Angst oder Freude kann ein Anwender dazu gebracht werden, gewünschte Aktionen auszuführen. Emotionen beeinflussen die Einschätzung von Gefahren und können so das Gefahrenbewusstsein herabsetzen. Dies kann ein Angreifer ausnutzen.

3.1.4 Wie kann der Schutz gewährleistet werden?

Der Browser muss einen Benutzer warnen, wenn er ein Täuschungsmanöver erkennt, oder dieses unschädlich machen. Gegen Beeinflussung der Emotionen ist ein Browser machtlos. Er muss immer davon ausgehen, dass ein Anwender auch gefährliche Inhalte aufruft, weil er dazu überlistet wurde. Aus diesem Grund muss er eine erste Prüfung der Anfrage, die er senden soll, durchführen. Nur wenn er diese als unverdächtig einstuft, sollte er sie auch durchführen. In einem zweiten Schritt muss die Antwort mit dem abgeglichen werden, was der Browser erwartet. Ist auch hier alles in Ordnung, kann er den Inhalt verarbeiten. Stößt er während der Verarbeitung auf etwas Verdächtiges, sollte er geeignete Maßnahmen ergreifen, z. B. den Inhalt oder bestimmte Funktionen blockieren.

Im Detail bedeutet dies:

Die Darstellung von (X)HTML ist zunächst ungefährlich. Sind aber z. B. Attribute gesetzt, die eine Aktion des Browsers erfordern, wie bei folgendem Tag: <META http-equiv="refresh" content="0; URL=…">, oder eine Ressource anfordern, wie bei <IMG src="…">, so ist Vorsicht geboten und der Browser sollte zuerst eine Überprüfung vornehmen, bevor er die Aktion ausführt oder eine weitere Anfrage stellt.

Genauso sollte auch immer die in der Adressleiste eingegebene URL validiert werden, um z. B. XSS-Attacken zu erschweren. Aber auch wenn vom Benutzer auf andere Weise, z. B. durch Absenden eines Formulars oder durch einen Klick auf einen Hyperlink, eine Ressource angefordert wird, sollte zunächst eine Sicherheitsüberprüfung der Anfrage stattfinden.

Besonders gefährlich sind aktive Inhalte. Sie versetzen einen Angreifer in die Lage, bestimmte Aktionen im Browser des Opfers hervorzurufen. Deshalb ist es wichtig, dass diese vom Browser überwacht werden. Manche Inhalte werden durch Plugins zur Verfügung gestellt und können somit schlecht überwacht werden. Andere, wie DOM und JavaScript, lassen sich allerdings sehr gut vom Browser begutachten. Aktive Inhalte dienen meist der Ergonomie einer Webseite und sind nicht von Grund auf gefährlich. Allerdings können sie meist auf benutzerspezifische Daten, wie Cookies oder Eingaben des Benutzers, zugreifen und somit auch potentiell Schaden anrichten. Daher ist es wichtig, dass der Browser unkritische Funktionen genehmigt und solche, die Schaden anrichten können, verweigert.

Plugins entziehen sich größtenteils der Überwachung des Browsers. Deshalb muss ein Browser auch dann eine Validierung vornehmen, wenn ein Plugin beispielsweise mittels Object-Tag angesprochen wird. Dabei ist es wichtig, ob die Seite, die solch einen Tag enthält, generell das Recht besitzt, das angesprochene Plugin zu laden und falls ja, ob auch die übergebenen Parameter für diese Domäne erlaubt sind. So kann der Browser beispielsweise überwachen, ob das Plugin eine URL der gleichen Domäne übergeben bekommt oder nicht und gegebenenfalls domänenübergreifende Zugriffe vermeiden.

Ebenfalls muss der Browser beim Cookie-Management eine Aufsichtsrolle übernehmen. Hiermit ist nicht gemeint, dass er die Cookies vor Zugriffen schützen muss, denn dies ist eine Grundvoraussetzung, die von einem Browser erwartet werden kann. Er sollte vielmehr auch überwachen, bei welchen Anfragen welches Cookie gesendet werden darf. Einer Anwendung sollte nicht erlaubt werden, ein Cookie gleichzeitig für http- und für https-Verbindungen zu nutzen. Da http-Verbindungen leicht ausgespäht werden können, kann ein Angreifer sonst das Cookie stehlen und somit zusätzlich Zugang zum https-gesicherten Teil der Applikation erhalten. Falls also ein Cookie

über eine http-Verbindung gegangen ist, sollte es bei https-Verbindungen nicht benutzt werden, um zu erzwingen, dass die Webanwendung für diese ein neues Cookie sendet. Andererseits dürfen Cookies, die über https-Verbindungen erhalten wurden auch nur über solche wieder zum Server übertragen werden. Des Weiteren sollte nicht browserweit ein gemeinsamer Cookie-Speicher verwendet werden, sondern eine Zuordnung zwischen Cookie und Browser-Ressource stattfinden. Gemeint ist hierbei nicht die Domäne der Webseite sondern das verwendete Tab oder Fenster des Browsers.

3.1.5 Gestaffelte Abwehr

Wichtig ist, dass man eine gestaffelte Abwehr aufstellt. Dies bedeutet, dass man auf die Wirkung einer Schutzmaßnahme nicht hundertprozentig vertraut, sondern dass sich Schutzfunktionen so ergänzen, dass immer noch eine greifen kann, falls eine andere den Angriff nicht abwehrt.

In der Praxis kann dies so aussehen: Der Browser soll DOM-basiertes XSS abwehren. Er filtert deshalb alle Anfragen, die er absendet, und verwirft diese, falls sie verdächtige Parameter enthalten oder gibt in solchen Fällen eine Warnung aus. Trotz dieser Maßnahme sollte der Browser die angeforderte Seite nicht als gefahrlos einstufen, da es einem geschickten Angreifer gelingen kann einen solchen Filter zu umgehen. Stattdessen sollte er, nachdem die Seite geladen ist, die Aktionsmöglichkeiten derselben so weit wie möglich einschränken. So sollten z. B. bestimmte JavaScript-Funktionen nur für wenige Seiten aufrufbar sein. Genauso sollten Plugins immer nur dann von einer Seite angesprochen werden können, wenn diese dafür eine Berechtigung hat. Außerdem könnte man selbst bei vertrauenswürdigen Seiten Maßnahmen vorsehen, so dass der Benutzer für kritisch erscheinende Aktionen eine Meldung erhält, z. B. falls das Objekt, auf das der Benutzer klickt, nicht für ihn erkennbar ist, weil es unsichtbar ist. So können Angriffe ggf. vom Anwender selbst, vom Browser beim Absenden der Anfrage und bei der Verarbeitung der Webseite durch den Browser erkannt werden. Erst wenn alle diese Maßnahmen versagen, ist der Angriff möglich.

3.1.6 Qualitätsmerkmale von Abwehrmaßnahmen

Wie hoch die Qualität einer Abwehrmaßnahme ist, hängt von vielen Faktoren ab, wie z. B. vom Anwender, von der Webanwendung, von der Umsetzung der Abwehrmaßnahme und dem Eingriffspunkt derselben. Je eher eine Abwehrmaßnahme greift, desto höher ist in der Regel auch ihre Qualität.

Nehmen wir an, eine Webseite enthält eine Clickjacking-Attacke. Klickt nun der Benutzer auf ein unsichtbares Objekt, so kann der Browser eine Warnung ausgeben und anzeigen, auf was der Anwender wirklich geklickt hat. Der Benutzer kann so noch vor Ausführung der durch diesen Klick ausgelösten Aktion entscheiden, ob dies wirklich seine Absicht war. Entscheidet sich der Benutzer richtig, wird der Angriff komplett verhindert. Trifft er die falsche Entscheidung und sendet beispielsweise eine Anfrage, kommt es darauf an, wie der Browser diese behandelt. Erkennt der Browser, dass die Anfrage versucht, eine verdächtige URL aufzurufen, kann er diesen Aufruf blockieren. Auch in diesem Fall wird der Angriff hundertprozentig abgewehrt. Bei den bisher genannten Maßnahmen kann der Angreifer keinen Schaden anrichten, sondern der Angriff wird aktiv blockiert. Erkennt der Browser die Gefährlichkeit der Anfrage nicht, wird eine neue Seite geladen, die z. B. unsichtbar für den Benutzer sein kann. Dann bleiben dem Browser nur noch Sekundärmaßnahmen zur Abschwächung des Angriffs. Soll z. B. eine CSRF-Attacke durchgeführt werden und der Benutzer hierzu automatisch angemeldet werden, kann der Browser dies erkennen, wenn die Seite versucht, automatisch beim Laden ein Formular abzusenden. Hierdurch kann er

immer noch die Folgen des Clickjacking-Angriffs verhindern, nämlich die CSRF-Attacke. Allerdings ist das Absenden der vorausgegangenen Anfrage, mit der der Hacker vielleicht auch weitere Informationen übertragen hat, nicht mehr rückgängig zu machen. Solche Sekundärmaßnahmen können i. d. R. also nur noch die Folgen eines Angriffs abschwächen, den Angriff selbst aber nicht mehr verhindern. Manche Filterfunktionen müssen auch zu Gunsten der besseren Bedienbarkeit Kompromisse eingehen, wodurch zusätzlich die Qualität dieser Abwehrmaßnahmen sinkt.

Primärmaßnahmen greifen immer dann, wenn die Aktion vom Benutzer ausgeht oder eine automatische Funktionalität des Browsers anspricht. Bei Aktionen, denen bereits ein Ereignis vorausgegangen ist, greifen normalerweise nur noch Sekundärmaßnahmen.

3.1.7 Zusammenfassung

Angriffsziel	Erreichbar durch	Umsetzbar mit Hilfe von	Abwehrmaßnahmen
- Link anklicken - Formular absenden	- Täuschen des Anwenders	- Clickjacking - Social Engineering	- Warnungen bei erkannten Täuschungsmanöver
- Automatischer Aufruf einer URL - automatischer Versand eines Formulars	- Nutzen von aktiven Inhalten - Nutzen von Tags mit Attributen zur Angabe einer URL	- CSRF - GIFAR	- Filtern von aktiven Inhalten - URL vor Absenden der Anfrage prüfen
- Klauen des Cookies - Klauen von Daten	- Absenden von selbst erzeugten Anfragen mit Daten - Nutzen eingeschleuster Objekte	- XSS in Verbindung mit Clickjacking, Social Engineering oder CSRF - GIFAR	- Daten nur in URLs und Formulare mit gleicher Domäne einfügen - Ziele von Anfragen einschränken

Tabelle 3-1: Abwehrmaßnahmen – Zusammenfassung

3.2 Allgemeine Sicherheitseinstellungen

Da eine Webanwendung immer Sicherheitslücken enthalten kann, muss ein Browser jegliche Inhalte als potentiell gefährlich ansehen. Vor allem, wenn sich die Domänen der verschiedenen Inhalte einer Seite unterscheiden, ist höchste Vorsicht geboten. Ein Angreifer kann auf einer Webseite, die er kontrolliert, jeden beliebigen Inhalt einfügen. So ist es auch möglich vom Benutzer als vertrauenswürdig angesehene Inhalte einzubinden um diese zu manipulieren. Eine weitere Möglichkeit ist, dass ein Angreifer es schafft, eine Webanwendung zu vergiften und so gefährliche Inhalte von einer von ihm kontrollierten Domäne einzubinden. Deshalb sollte der Browser Inhalte, die von einer anderen Domäne stammen standardmäßig nicht laden. Zu diesen Objekten gehören beispielsweise IFrames, Frames, Objects, Applets. Wenn nun der Browser auf solch ein Objekt stößt, sollte er einen Platzhalter anbieten, den man anklicken kann, so dass der Benutzer entscheiden kann, ob er den Inhalt anzeigen lassen möchte oder nicht. Allerdings ist es sinnvoll, dass der Browser dies erst nach einer Warnung zulässt. Des Weiteren sollte es möglich sein, Ausnahmen zu definieren. So sollten zum einen Inhalte vom Anwender bestimmt werden können, die auf allen Webseiten angezeigt werden können, z. B. zur Einbindung von Anwendungen wie Google-Maps. Andererseits sollten Ausnahmeregelungen für bestimmte Domänen anlegbar sein, so dass beispielsweise Subdomänen von Internetseiten großer Konzerne Inhalte anderer Subdomänen desselben Konzerns einbinden können.

Zusätzlich sollte der Browser die Möglichkeit bieten, Rechte auf Level von Webseiten zu vergeben. So sollte der Benutzer für jede Seite festlegen können, ob diese JavaScript oder bestimmte Plugins benutzen darf, ob sie IFrames und Frames enthalten darf, ob sie in einem Frame angezeigt werden darf, usw. Ist vom Benutzer kein Recht vergeben, sollte eine Standartregelung greifen. Um die Handhabung einfacher zu gestalten, könnte man eine oder mehrere zentral gepflegte Listen anbieten, die regelmäßig aktualisiert werden. Natürlich sollte der Benutzer entscheiden können, ob er eine Liste integrieren möchte und wenn ja, welche. Es wäre außerdem sinnvoll, bestimmte Webseiten in Gruppen zusammenfassen zu können, um die Freigaben zu erleichtern. Die Regeln des Benutzers sollten dabei immer Vorrang vor den zentral gepflegten Regeln haben. Falls aber keine vom Anwender festgelegte Regel für eine Webseite existiert, sollten zunächst die zentral gepflegten Regeln überprüft werden. Erst wenn auch hier keine zutrifft, sollte die Standardrichtlinie verwendet werden. Wenn eine Webseite Mitglied in verschiedenen Gruppen auf der gleichen Prioritätsstufe ist, sollte immer die striktere Regel verwendet werden. Ein Beispiel hierzu:

Die zu überprüfende Webseite ist Mitglied in folgenden Gruppen:

- Gruppe A: IFrames erlauben; Applets erlauben; PDFs als Object einbinden erlauben
- Gruppe B: JavaScript erlauben; bei IFrames nur „www.example.com“ erlauben, andere verbieten
- Gruppe C: Applets verbieten

So sollte die Webseite in Summe folgende Rechte erhalten: PDFs als OBJECT einbinden erlauben, JavaScript erlauben und IFrames von „www.example.com“ erlauben

Solch eine Regelung kann ein hohes Maß an Sicherheit bieten, da die meisten Angriffe darauf beruhen, Elemente verschiedener Domänen zusammenzuführen oder aufzurufen. Leider geht diese Sicherheit zu Lasten der Benutzbarkeit, allerdings kann dieses Problem durch zentral gepflegte Listen teilweise behoben werden. Je besser diese Listen sind, desto mehr Benutzer könnten sich mit einem solchen Modell anfreunden. Außerdem könnten die Webseitenbetreiber selbst Listen zur Verfügung stellen, da diese genau wissen, welche Inhalte ihre Seite anzeigen sollte und welche unerwünscht sind. Deren Listen sollten aber nur Regeln für ihre eigenen Seiten enthalten dürfen, um Manipulationen anderer Domänen auszuschließen.

Für die Akzeptanz von Sicherheitsmechanismen ist wichtig, dass der Großteil der vom Anwender benutzten Seiten weiterhin funktioniert. Außerdem sollte eine einfache Anpassbarkeit an die eigenen Bedürfnisse möglich sein. Dies kann durch Frontends erreicht werden, die die Komplexität der Regeln für Benutzer verstecken und verständliche, vorgefertigte Regeln für unkundige Benutzer anbieten, aber auch technisch versierte Anwender die volle Funktionalität ausnutzen lassen.

3.3 Sichern von Eingaben des Benutzers

Der Browser muss sicherstellen, dass Eingaben des Benutzers nicht für Angriffe verwendet werden können. Hierzu ist es wichtig, dass der Benutzer genau merkt, wo er seine Eingaben tätigt. Außerdem müssen aktive Inhalte überwacht werden, die auf Benutzereingaben zugreifen wollen. Man kann die Sicherung der Benutzereingaben in zwei Gebiete aufgliedern. Zum einen ist dies die Sicherung der Benutzeraktionen, wie Mausklicks und Tastatureingaben und zum anderen der Schutz der bereits getätigten Daten in Eingabefeldern. Eine spezielle Rolle beim Datenschutz spielt die Absicherung von automatisch eingefügten Anmeldedaten.

3.3.1 Sicherung der Benutzeraktionen

Der Browser muss gewährleisten, dass der Benutzer einen vertrauenswürdigen Pfad für seine Eingaben erhält, d. h. es muss sichergestellt werden, dass die Eingaben ohne Änderung an der richtigen Stelle landen. Dies ist meist nicht so einfach, wie es klingt. Gibt ein Anwender Daten in ein Formular ein und schickt diese ab, kann der Browser nicht wissen, ob der Benutzer nicht vielleicht überlistet wurde. Er kann z. B. seine Anmeldedaten in ein durch XSS gefälschtes Anmeldeformular eingegeben haben, das die Daten beim Absenden nicht an die vom Opfer assoziierte, sondern an eine vom Angreifer kontrollierte Seite schickt. Deshalb sollte der Browser eine Überprüfung des abzusendenden Formulars vornehmen um sicherzustellen, dass die Möglichkeit der Fälschung möglichst gering ist. Wie diese Überprüfung aussehen sollte, ist im Abschnitt [Überprüfen von Formular-Zielen – Seite 45] beschrieben. Klickt ein Benutzer auf ein Objekt, kann dies bestimmte Ereignisse auslösen. Ist das Objekt beispielsweise ein Hyperlink, so setzt der Browser normalerweise eine Anfrage ab. Bei einem Options- oder Kontrollfeld ändert er i. d. R. dessen Status und im Fall eines Absende-Buttons sendet er ein Formular ab. Alle Elemente können zusätzlich Event-Handler aufrufen. Dies sind meist JavaScript-Funktionen. Da JavaScript auch auf das DOM Zugriff hat, kann es somit die dargestellte Seite nahezu beliebig manipulieren. Dies wird auch bei Clickjacking genutzt. Hierbei wird ein meist unsichtbares Objekt während des Klickens über ein anderes Objekt gelegt, um somit den Klick des Benutzers umzulenken. Der Anwender kann so, ohne sein Wissen, unter anderem ein Formular absenden oder Einstellungen verändern. Um dies zu verhindern, sind zwei verschiedene Sicherheitsfunktionen nötig. Zum einen sollten JavaScripts überwacht werden: Mit einem geeigneten Filter können so Angriffe verhindert werden. Wie solche Filter aussehen können, ist im Abschnitt [JavaScript-Filter – Seite 42] zu lesen. Außerdem sollte der Browser immer sicherstellen, dass der Benutzer sieht, was er anklickt. Überlagern sich Objekte an der Stelle, wo der Anwender geklickt hat, muss der Browser sicherstellen, dass der Benutzer auf das von ihm gewünschte Objekt geklickt hat. Kann er dies nicht, muss er eine Warnung ausgeben. Nachfolgend ist erklärt, wie eine Überprüfung hierzu aussehen sollte. Bei sich überlagernden Objekten muss der Browser ermitteln, ob das Objekt, auf das der Benutzer geklickt hat

- für den Benutzer sichtbar ist,
- eine Mindestgröße erreicht hat (hierbei ist der sichtbare Teil zu beachten),
- ob es von umgebenden Objekten unterscheidbar ist und
- ob das Objekt erst kurz vor dem Klick für den Benutzer sichtbar wurde oder schon länger sichtbar war (Reaktionszeit).

Die Sichtbarkeit eines Objekts lässt sich leicht vom Browser ermitteln, da er für das Zeichnen der sichtbaren Objekte verantwortlich ist. Auch die Größe des sichtbaren Teils lässt sich leicht berechnen und somit mit einer Mindestgröße vergleichen. Noch dazu kennt der Browser den Zeitpunkt, wann das Objekt gezeichnet wurde und kann somit entscheiden, ob die Anzeigedauer ausreichend ist. Die Unterscheidung von umgebenden Objekten stellt sich allerdings komplex dar. Zwar kann der Browser die Farben des Objekts mit seiner Umgebung vergleichen und hieraus einen Schluss ziehen, ein Angreifer kann eine Seite aber auch so zusammenbauen, dass sich Objekte anderer Domänen, wie eine weitere Seite - dargestellt in einem IFrame - perfekt in das Bild einfügen. Deshalb sollte der Browser darüber hinaus überprüfen, ob die umgebenden Objekte und das angeklickte Objekt zur selben Domäne gehören, siehe hierzu auch [Allgemeine Sicherheitseinstellungen – Seite 34].

Das Austricksen eines Benutzer ist auch durch andere Täuschungsmanöver möglich. So kann eine gefährliche Webseite beispielsweise dem Anwender bekannte Elemente simulieren und ihn so zu

einem Klick bewegen. Da die Browseroberfläche immer mehr mit der angezeigten Webseite verschmilzt, wird die Fälschungsmöglichkeit immer größer. So wird z. B. eine Warnung des Microsoft Internet Explorers am oberen Rand des Anzeigebereichs einer Webseite eingeblendet, obwohl dieser eigentlich für die Darstellung von Webinhalten gedacht ist und nicht für Meldungen des Browsers, siehe [Abbildung 3-3: IE mit Warnung (unsicher)]. Nachfolgend ist ein Screenshot zu sehen, der eine solche Warnung zeigt.

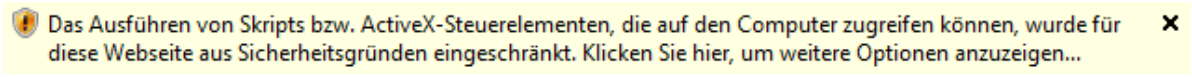

Abbildung 3-1: Sicherheitswarnung des Internet Explorer

Im Mozilla Firefox gibt es ähnliche Warnungen, die auf die gleiche Weise angezeigt werden. Außerdem verwenden viele Plugins eine Art Statusleiste, die oft oberhalb der normalen Browserstatusleiste im Darstellungsbereich der Webseite angezeigt wird und leicht fälschbar ist. Nachfolgend wird eine Statusmeldung des NoScript-Plugins gezeigt.

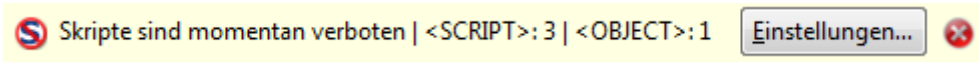

Abbildung 3-2: Statusmeldung des NoScript-Plugins für den Firefox

Solche Elemente sind für Angreifer leicht nachzubilden. Wenn sie diese nun in ihre Seite einbauen und einem unbedarften Benutzer präsentieren, wird dieser vielleicht aus Gewohnheit oder Unwissenheit auf diese Leiste klicken, da er sie für ein Browserelement hält. Überprüft nun der Browser alle Kriterien, die sicherstellen, dass der Anwender bewusst auf ein Element geklickt hat, so stellt er nichts Auffälliges fest. Dennoch wurde der Benutzer getäuscht. Die Verantwortung ist hierbei zum Großteil dem Browser zuzuschreiben, der dem Benutzer keinen sicheren Pfad zu der vom Browser erstellten Meldung bietet. Dabei wäre dies keine große Angelegenheit. Die Meldung müsste lediglich so dargestellt werden, dass sie nicht durch eine Seite nachgebildet werden kann. Am einfachsten ist dies möglich, indem die Meldung nicht direkt im Darstellungsbereich zu sehen ist. Stattdessen muss zwischen der Meldung und dem Darstellungsbereich ein graphisches Element eingefügt werden, das diesen von der Meldung sichtbar abgrenzt. Außerdem darf dieses graphische Element nicht von einer Seite nachbildbar sein. Dies kann dadurch erreicht werden, dass das Element immer angezeigt wird, auch wenn keine Warnung vorhanden ist. So ist dem Anwender klar, dass der Darstellungsbereich der Seite immer unter diesem Element beginnen muss. In der Theorie hört sich das viel komplizierter an, als es ist. Deshalb werde ich dies an einem Beispiel illustrieren. Ich werde hierzu die Warnung des Internet Explorers verwenden.

Die ersten beiden Bilder zeigen den Internet Explorer mit Meldung und ohne (nach dem Klick auf die Meldung und Bestätigung der folgenden Dialoge).

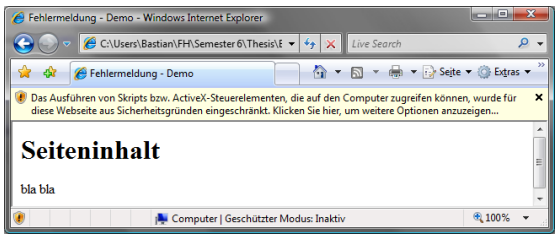

Abbildung 3-3: IE mit Warnung (unsicher)

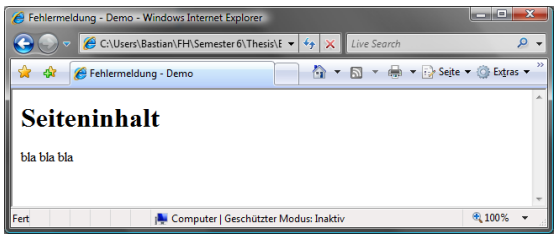

Abbildung 3-4: IE ohne Warnung (unsicher)

Die nächsten beiden Bilder weisen zusätzlich einen roten Balken auf. Dieser grenzt den Darstellungsbereich ab. So wäre es dem Nutzer möglich, eine gefälschte Meldung daran zu erkennen, dass der rote Balken oberhalb der Meldung zu sehen ist.

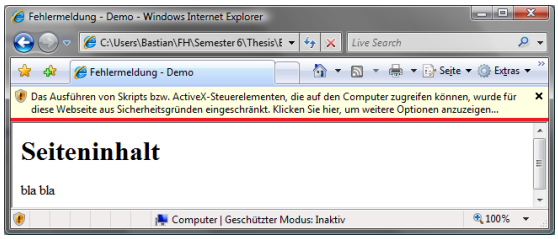

Abbildung 3-5: IE mit Warnung (sicher)

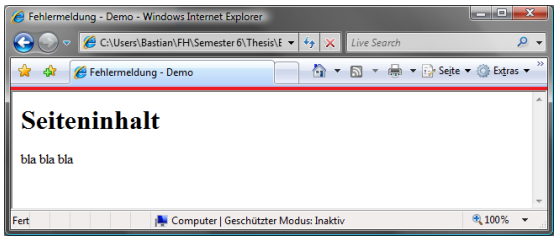

Abbildung 3-6: IE ohne Warnung (sicher)

Natürlich könnte das Design des Trennungselements noch besser an die gesamte Browseroberfläche angepasst werden, so dass es nicht störend wirkt. Dieses Beispiel soll nur verdeutlichen, wie man dem Benutzer sichtbar machen kann, ob eine Meldung vom Browser selbst oder von der Webseite stammt. Diese Lösung erhebt keinen Anspruch auf attraktives Design.

Falls der Anwender dennoch aus Unwissenheit auf ein gefälschtes Element klickt, kann der Browser dies durch die bereits genannten Verfahren nicht erkennen. Allerdings sollte er weitere Maßnahmen ergreifen, um zu verhindern, dass der Klick für den Benutzer schwere Folgen hat. Zum einen sind dies Schritte zum Abschwächen gefährlicher Aktionen durch JavaScript-Filter verbunden mit Abwehrmaßnahmen gegen gefährliche Anfragen. Zum Anderen ist das die Gewährleistung des Schutzes der Daten des Benutzers.

3.3.2 Datenschutz

Wie kann der Browser die Benutzerdaten schützen? Hierzu müsste er wissen, wann und wohin er die Daten senden darf. Dies kann von Webanwendung zu Webanwendung unterschiedlich sein, was diesen Vorgang nicht möglich macht, oder doch?

Um einen optimalen Schutz der Daten gewährleisten zu können, müsste der Browser tatsächlich auf eine Webanwendung speziell zugeschnitten sein. Dies ist natürlich nicht möglich. Jedoch kann man durch Beachtung einiger Regeln die Sicherheit wesentlich erhöhen.

Für einen Angreifer gibt es zwei unterschiedliche Ziele in Bezug auf die Benutzerdaten. Einerseits besteht der Missbrauch der Daten, ohne dass der Hacker diese selbst kennt. Dies findet vor allem bei der sogenannten CSRF-Attacke und bei ähnlichen Angriffen Verwendung. Andererseits existiert die Gefahr des Ausspionierens der Daten, so dass der Angreifer diese für sich selbst nutzen kann.

3.3.2.1 Schutz gegen Datenmissbrauch

Um Datenmissbrauch begehen zu können, wird dem Opfer meist auf irgendeine Weise ein gefährlicher Link untergeschoben. Dieser stammt allerdings meist aus einer externen Anwendung, wie einem Email Programm oder von einer vom Angreifer kontrollierten Seite. In beiden Fällen kann der Browser Einschränkungen vornehmen. Einerseits, wenn ein Link von einem externen Programm aufgerufen wird und andererseits, wenn sich Quell- und Zieldomäne des Link unterscheiden. Eine Ausnahme hierbei ist, wenn der Angreifer es schafft, durch eine Attacke einen gefälschten Link direkt in die anzugreifende Webanwendung einzuschleusen. In diesem Fall stehen dem Browser keine Informationen zur Verfügung, die ihn den Link als schädlich erkennen lassen.

Falls die Webanwendung eine Anmeldung des Benutzers erfordert, bietet dies einen hohen Schutz gegen Datenmissbrauch. Nun werden viele der Meinung sein, dass dies für CSRF kein Hindernis darstellt, da dieser Angriff auch solche Hürden überwinden kann. Dies liegt allerdings daran, dass die Browser derzeit noch keine geeigneten Schutzmaßnahmen implementieren. Um CSRF nahezu unmöglich zu machen, muss ein Browser den Anmeldevorgang, i. d. R. das Abschicken eines Formulars, und die authentifizierte Verbindung, normalerweise durch ein Cookie signalisiert, besser schützen.

Speichert der Browser die Anmeldedaten eines Benutzers, muss der Anwender meist nur noch das Anmeldeformular abschicken, indem er z. B. auf einen Login-Button klickt, da Benutzername und Kennwort bereits vom Browser eingefügt wurden. Dies stellt für CSRF-Angriffe natürlich eine große Erleichterung dar. Eine gefährliche Webseite muss dem Browser nur ein Anmeldeformular präsentieren, das zu der Webanwendung gehört, die der Hacker angreifen will. Außerdem kann dieses Formular automatisiert abgeschickt werden. Wenn der Browser die Anmeldedaten einfügt, ist das Opfer somit eingeloggt. Diese Möglichkeit sollte vom Browser verhindert werden. Aber auch wenn dies unterbunden wird, kann der Angreifer den Benutzer auf andere Weise, beispielsweise durch Social Engineering, dazu bringen, sich in der Webanwendung einzuloggen. Ist dies erst einmal geschehen, erhält der Anwender normalerweise ein Session-Cookie, um sich nicht andauernd bei der Webanwendung neu anmelden zu müssen. Wenn der Angreifer dies geschafft hat, muss er dem Opfer nur noch einen präparierten Link unterschieben, der die vom Hacker gewünschte Aktion zur Folge hat, z. B. die Deaktivierung der Routerfirewall. Würde der Browser ein besseres Cookie-Management anwenden, wäre solch ein Angriff nicht so leicht durchführbar.

3.3.2.1.1 Anmeldedatenschutz

Um Anmeldedaten zu schützen, sollte ein Browser sicherstellen, dass diese immer vom Benutzer abgesendet werden, wodurch automatische Anmeldungen geblockt werden können. Da der Browser die Eingaben des Anwenders erfasst, kann er auch erkennen, ob das Formular nun von ihm oder automatisiert durch die Webseite abgeschickt werden soll. So sollte er Formulare, die Passwortdaten enthalten (INPUT-Tag mit type-Attribut „password“) nur absenden, wenn dies vom Benutzer explizit gefordert wurde. Falls das Formular hierbei zusätzlich einen Event-Handler aufruft, sollte dieser selbst keine weiteren Anfragen absenden dürfen. Außerdem müssen natürlich auch die üblichen Einschränkungen für Formulare gelten, siehe [Request-Filter (URL-Filter und Form-Filter) – Seite 44]. Des Weiteren sollte ein Browser immer versuchen, Anmeldeformulare über https abzusenden, auch wenn dies nicht so in der Action angegeben wurde. Falls das Formular über http gesendet werden soll und eine Verbindung zum Server über https fehlschlägt, könnte der Browser einen Fallback auf http unterstützen. Am besten wäre es, wenn dies der Benutzer durch Einstellungen festlegen kann. Außerdem müsste der Browser eine Warnung ausgeben, wenn das Anmeldeformular von einer Domäne abgeschickt werden soll, die sich von der Zieldomäne unterscheidet. Andernfalls kann der Angreifer sonst vielleicht die Anmeldedaten abhören, wenn diese nicht über eine https-, sondern über eine http-Verbindung gesendet werden.

3.3.2.1.2 Cookie-Schutz

Die Cookies sollten durch den Browser möglichst gut geschützt werden, da sie oft der Authentifizierung dienen. Eine Maßregel ist es, Cookies, die über http gesendet wurden, nur für http-Verbindungen zu verwenden und andere, die über https gesendet wurden, nur über https zu senden. Außerdem sollte ein Browser eine Einstellungsmöglichkeit bieten, um http-only Cookies zu erzwingen. Dies hat den Vorteil, dass Webanwendungen, bei denen dies realisiert ist, gegenüber

Angriffen, bei denen über JavaScript oder Plugins auf Cookies zugegriffen wird, z. B. auch bei GIFAR, weniger anfällig sind. Leider ist dies nicht immer möglich, da einige Webanwendungen dadurch nicht mehr funktionieren würden.

Um Angriffe abzuwehren, bei denen authentifizierte Verbindungen durch andere Webseiten mitbenutzt werden, kann ein Browser die Cookies pro Tab bzw. Fenster verwalten. Ist nun ein Benutzer beispielsweise in einem Tab auf dem Webinterface des lokalen Routers eingeloggt und ruft in einem anderen Tab eine gefährliche Webseite auf, die eine CSRF-Attacke durchführen möchte, kann diese die authentifizierte Verbindung des anderen Tabs nicht mitbenutzen, da für das Tab, in dem der Angriff stattfindet, das hierzu nötige Cookie unsichtbar ist. Somit könnte ein Angriff nur in dem Tab durchgeführt werden, in dem der Anwender die authentifizierte Verbindung aufgebaut hat. Allerdings muss der Browser auch sicherstellen, dass das Cookie verworfen wird, sobald der Benutzer eine andere URL mit diesem Tab aufruft, da sonst die nachfolgende Seite einen Angriff ausführen könnte.

3.3.2.2 Schutz gegen Datenklau

Um Daten klauen zu können, muss der Angreifer seinem Opfer einen Code unterschmuggeln, der auf die gewünschten Daten Zugriff hat, und dann diese dem Hacker zukommen lassen kann. So kann z. B. meist ein Cookie entführt werden, indem ein Angreifer die Cookie-Daten per JavaScript ausliest und dann den Browser eine Anfrage ausführen lässt, in welcher er diese Daten als URL-Parameter anhängt. Hierbei missachtet der Browser eine grundlegende Sicherheitsregel. Er sollte keine Daten ungeprüft an beliebige Domänen versenden. Bei den meisten Angriffsszenarien ist der Schutz von Benutzerdaten einfach zu implementieren. Immer dann sollte das Versenden von Daten verwehrt werden, wenn der Browser die Daten an eine andere Domäne senden soll, als die, aus der die Daten stammen. Leider jedoch führen die Browser derzeit eine solche Überprüfung nicht durch. Lediglich wenn der Angreifer es schafft, ein Objekt zum Sammeln der Daten in der gleichen Domäne einzuschleusen, kann er diese Methode der Überprüfung umgehen.

Wie sollte nun eine Überprüfung im Detail aussehen? Eigentlich gibt es schon längst ähnliche Mechanismen und zwar auf der Seite des Webservers. Einige Frameworks bieten einen so genannten Tainting-Mechanismus an. Bei diesem werden alle Objekte, die von außerhalb eintreffen als „tainted“ (mit einem Makel behaftet) markiert. Wenn nun der Webserver eine gefährliche Aktion mit diesen Daten durchführen soll, z. B. eine Interpretierung dieser Zeichenkette durch eine eval-Funktion, wird dies verweigert. Alle Objekte, die eine entsprechende Markierung aufweisen, werden als potentiell gefährlich angesehen. Einen ähnlichen, etwas erweiterten Mechanismus könnte man nun auch im Browser einsetzen. Man könnte alle URL-Parameter als „tainted“ markieren, da diese auch Angriffe wie DOM-basiertes XSS enthalten können. Als zweite Markierung sollten alle vom Benutzer eingegebenen Werte bzw. vom Browser automatisch eingefügten Daten sowie Cookies als „privat“ angesehen werden. Neben dieser Markierung sollte auch vermerkt werden, zu welcher Domäne diese Daten gehören. Nehmen wir an, ein Benutzer gibt Daten in ein Formular ein und sendet dieses ab. Weiterhin enthält das Formular einen Event-Handler, der beim Absenden aufgerufen wird. Dies ist nichts Ungewöhnliches, da die Formulardaten beispielsweise validiert werden können. Greift nun der Event-Handler auf die Formulardaten zu, sollte der Browser jede Variable, bei denen die Daten mit einfließen, als „privat“ markieren. Ist diese Variable bereits als „privat“ markiert und unterscheiden sich die Domänen, so sollte der Browser die Aktion abbrechen. Dies ist jedoch nicht das Einzige, auf was der Browser achten muss. Versucht irgendeine Funktion als „privat“ markierte Daten in eine URL zu integrieren oder in einem Formularfeld einzufügen, so muss

der Browser immer Quell- und Zieldomäne überprüfen. Nur bei einer Übereinstimmung darf die Aktion durchgeführt werden. Außerdem muss darauf geachtet werden, dass sich die Domäne der URL oder des Ziels des Formulars nicht mehr verändern darf, wenn Benutzerdaten darin eingeflossen sind. Darüber hinaus sollten als „tainted“ markierte Daten nie in Funktionen wie eval einbezogen werden. So kann die Sicherheit im Browser wesentlich erhöht werden. Denn selbst wenn der Angreifer es schafft, die Webanwendung zu manipulieren, kann er so i. d. R. nicht an Benutzerdaten gelangen. Außerdem ist die Funktion der Webanwendung selbst nicht beeinträchtigt: Sie kann weiterhin Daten validieren, ändern oder auf andere Weise verarbeiten, z. B. wenn ein Passwort nicht im Klartext übermittelt werden soll, so lange die Domänengrenze nicht überschritten wird.

Durch solch eine strikte Einschränkung können leider auch Probleme mit Webseiten entstehen, bei denen unterschiedliche Domänen zusammenspielen. Dies ist bei großen Unternehmen meist der Fall. Deshalb sollte es dem Benutzer möglich sein, für bestimmte Webseiten Ausnahmen festzulegen. Hierzu könnte man zwei Listen führen. Zum einen eine Liste mit Gruppen von Domänen, die untereinander kommunizieren dürfen und zum anderen eine Liste mit Mustern und Regeln für die eine Kommunikation erlaubt ist. Nachfolgend werde ich zur Illustration Beispiele anführen.

Beispiel für Gruppen von Domänen, die untereinander kommunizieren können:

- Gruppe 1:
  - de.example.com
  - en.example.com
  - img.example.com
- Gruppe 2:
  - www.example.com
  - www.example.org

Im angeführten Beispiel können die Domänen „de.example.com“, „en.example.com“ und „img.example.com“ untereinander kommunizieren. Auch die Verständigung zwischen „www.example.com“ und „www.example.org“ ist möglich.

Beispiel für Domänen mit besonderen Regeln:

- *.example.net: *.example.net
- *.example.com, *.example.org: *.example.com

Im aufgezeigten Beispiel ist es allen Subdomains von „example.net“ möglich, untereinander zu kommunizieren. Außerdem ist es allen Subdomains von „example.com“ und „example.org“ erlaubt, sich mit Subdomains von „example.com“ auszutauschen. Da dies eine Art von Filterung ist, sollten die Akzeptanzkriterien von Filtern beachtet werden. Diese werden im nächsten Abschnitt besprochen.

3.4 Filter

Bei der Filterung ist es wichtig, dass man, wenn möglich, immer ein Whitelisting verwendet. Bei diesem wird grundsätzlich zunächst alles verboten und nur das erlaubt, was man explizit angegeben hat. Vergisst man eine Funktion oder wird der Funktionsumfang erweitert, werden die vergessenen bzw. neuen Funktionen zuerst einmal geblockt. Dies hat den Vorteil, dass ein Angreifer keinen Nutzen daraus ziehen kann, wenn man nicht an alles gedacht hat, oder wenn neue, vielleicht

gefährliche Funktionen eingebaut, aber noch nicht erfasst wurden. Der Nachteil liegt darin, dass auch ein Anwender die neue Funktionalität nicht nutzen kann, somit leidet die Ergonomie darunter.

Damit Filter von einem Anwender akzeptiert werden, müssen sie deshalb möglichst auf dem neuesten Stand der Technik sein. Außerdem sollten sie an die Bedürfnisse des Benutzers anpassbar sein. Kann ein Anwender bestimmte benötigte Funktionen nicht nutzen, so wird der Filter von ihm abgeschaltet und ist somit nutzlos. Leider ist es bei vielen Filtern so, dass sie entweder einfach zu bedienen sind oder einen großen Funktionsumfang bieten. Beides ist suboptimal. Ist ein Filter so entworfen, dass er von jedem bedient werden kann, deckt er zumeist auch keine komplexen Funktionen ab. Solche einfache Filter benötigen meist nur einen Klick zum Ändern einer Option, z. B. JavaScript aktivieren oder deaktivieren. Bietet ein Filter andererseits einen großen Funktionsumfang und ist durch Ausdrücke steuerbar, die für Profis mächtige Werkzeuge darstellen, so ist er von einem normalen Anwender meist nicht mehr bedienbar.

Deshalb sollten Filter so gestaltet werden, dass sie beide Gruppen ansprechen. Dies ist dadurch möglich, dass die Filter mit Ausdrücken arbeiten können und so für fortgeschrittene Anwender viele Einstellungsmöglichkeiten bieten. Die Ausdrücke für Filtereinstellungen sollten aber in Frontends zunächst durch einfache, auswählbare Elemente versteckt werden. Dennoch sollte das Frontend dynamisch gestaltet werden können, so dass es auch den Umstieg zwischen professioneller und einfacher Ansicht leicht ermöglicht. Es sollte mehrere Konfigurationsstufen bieten, bei denen sich die Einstellungsmöglichkeiten des Anwenders immer weiter vergrößern. Für Profis sollte es auch möglich sein, direkt vom Filter verarbeitbare Ausdrücke angeben zu können.

Um sicherzustellen, dass auch unkundige Benutzer mit qualitativ hochwertigen Regeln versorgt werden, können Filterregelsätze zentral gepflegt werden und über eine Art Abonnement durch Benutzer integrierbar gemacht werden. Solche von Profis erstellten Regeln ermöglichen es dem technisch Unkundigen, in den Genuss hochqualitativer Filterregeln zu kommen. Da viele Anwender mit Standardeinstellungen arbeiten oder diese nur gering abändern, sollten solche Listen automatisch integriert sein, falls dies nicht explizit anders gewünscht wird. Es sollte dem Benutzer aber auch möglich sein, diese Abonnements wieder zu löschen. Es ist wichtig, dass vom Benutzer erstellte Regeln Vorrang vor den Regeln zentral gepflegter Listen haben. Die Standardrichtlinien sollten möglichst große Einschränkungen vornehmen aber erst greifen, wenn kein Filterkriterium der benutzerdefinierten Regeln oder zentral gepflegten Listen greift.

Um die Verwaltung bestimmter Regelsätze zu erleichtern, sollte es möglich sein, Gruppen zu definieren und die zu überprüfenden Elemente verschiedenen Gruppen beitreten zu lassen. Regeln, die etwas verweigern, sollten dabei Vorrang vor den anderen Regeln haben, wenn sie auf der gleichen Prioritätsstufe liegen.

3.4.1 JavaScript-Filter

Eine wichtige Voraussetzung für hochqualitative JavaScript-Filter wurde bereits in Kapitel [Schutz gegen Datenklau – Seite 40] angesprochen, nämlich der sogenannte Tainting-Mechanismus. Alle Parameter, die mit der URL zu tun haben, sollten als „tainted“ gekennzeichnet werden, wie z. B. das location Objekt oder document.URL. Alle Benutzereingaben sowie Cookies sollten als „privat“ markiert werden, mit einem Vermerk zu ihrer Herkunft.

3.4.1.1 Einstellungsmöglichkeiten

JavaScript-Filter sollten verschiedene Einstellungsmöglichkeiten bieten. Für jedes Objekt, jede Eigenschaft und jede Methode sollte einstellbar sein, ob Variablen verarbeitet werden dürfen, die als „tainted“ gekennzeichnet sind. Analog gilt dies auch für als „privat“ markierte Variablen. Zudem sollte es möglich sein, festzulegen, dass die markierten Variablen nur escaped verarbeitet werden dürfen. Hierbei sollten Zeichen ersetzt werden, die eine bestimmte Bedeutung in HTML haben. Die folgenden Ersetzungen sind dazu durchzuführen: „&“ (Kaufmannsund) durch „&“, „<“ (kleiner als) durch „<“, „>“ (größer als) durch „>“, „"“ (doppelte Anführungszeichen) durch „"“ und „'“ (einfaches Anführungszeichen) durch „'“. Es sind auch andere Escaping-Mechanismen denkbar, je nachdem wie die Parameter verarbeitet werden. Ein HTML-Escaping ist beispielsweise für document.write() sinnvoll. Werden Daten an Plugins weitergegeben, müssen ggf. andere Regeln beachtet werden. Außerdem sollte einstellbar sein, ob das Objekt bzw. die Eigenschaft oder Methode überhaupt zugreifbar sein sollte oder ob der Zugriff generell verwehrt werden soll. Des Weiteren sollte der Kontext festgelegt werden können, in dem diese Einstellungen gültig sind. So kann es sinnvoll sein, dass als „privat“ gekennzeichnete Variablen von einer Funktion verarbeitet werden können, solange die Verarbeitung innerhalb des eigenen Bereichs bleibt, z. B. solange die Funktion nicht von einem Applet aufgerufen wird. Hierzu muss eine Liste von Aufrufern geführt werden, was vom Browser zu bewerkstelligen ist. Es ist empfehlenswert, Filterregeln als Ausdrücke mit einer gewissen Syntax zu speichern. So kann man diese erweitern, falls weitere, bisher nicht bedachte Funktionen von Nöten sind.

3.4.1.2 JavaScript deaktivieren

JavaScript zu deaktivieren, wie es zumeist gefordert wird, kann auch große Nachteile bezüglich der Sicherheit mit sich bringen. So kann eine Webseite z. B. einen sogenannten Frame-Bursting-Code enthalten. Dieser ist dafür zuständig, dass die Webseite nicht in einem Frame dargestellt werden kann. Falls dies aber doch passiert, erkennt dies das JavaScript und lädt die Seite erneut im Topframe. Auch andere Sicherheitsmaßnahmen können mit JavaScript programmiert sein. So schickt der Browser Passwörter z. B. im Klartext. Ungeachtet davon, dass für Loginformulare https genutzt werden sollte, könnte eine Webanwendung die Passwörter zusätzlich verschlüsseln bzw. hashen, so dass diese z. B. auch nicht im Klartext in Logfiles auftauchen. Außerdem stellt diese zusätzliche Sicherung auch eine Stärkung des Schutzes von Daten bei Man-in-the-Middle-Attacken dar. Auch solche Sicherheitsmaßnahmen würden durch deaktiviertes JavaScript ausgehebelt werden.

Deshalb ist es meist sinnvoll, wenigstens ein paar JavaScript-Funktionen für alle Webseiten zuzulassen. Unbedenklich sind z. B. logische und arithmetische Funktionen. Beim Zugriff auf Benutzerdaten sollte JavaScript eingeschränkt werden, so dass es nur Daten lesen und schreiben kann, die derselben Domäne angehören. Damit ist auch gemeint, dass es keine privaten Daten in Formulare eintragen darf, welche als Action eine abweichende Domäne aufweisen. Befindet sich ein Formular oder eine URL in einem Frame, sollte die ganze Hierarchie durchlaufen werden, ob überall die Berechtigung besteht, Daten mit der angegebenen Domäne auszutauschen, wie dies bereits in [Schutz gegen Datenklau – Seite 40] erläutert wurde.

3.4.1.3 Filterempfehlungen

Es ist sinnvoll, bestimmte Funktionen zu blockieren. So sollte ein JavaScript möglichst nicht auf Cookies, URL-Daten und den Referer zugreifen können. Einer Webanwendung sind zur Übermittlung dieser Daten andere Möglichkeiten gegeben. So kann ein URL-Parameter einem JavaScript direkt als Parameter übergeben werden, oder in einer Variablen gekapselt. Dies hat den Vorteil, dass Angriffe

mit gefälschten URLs nicht so leicht zum Ziel führen. Außerdem sollte JavaScript nicht den z-Index eines Elements ändern können, das eine andere Domäne als Quelle besitzt.

Ein weiterer Vorteil ist, dass spezielle JavaScript-Filter für bestimmte Webanwendungen entwickelt werden können, um diese optimal einzuschränken. Denn je weniger eine Anwendung clientseitig darf, desto schwieriger ist es auch, Angriffe im Browser durchzuführen, selbst wenn ein Hacker die Webanwendung serverseitig vergiftet hat oder sie durch gefälschte URL-Parameter beeinflussen will. Solche Einschränkungen können von Profis vorgenommen werden, da das JavaScript einer Anwendung im Quelltext vorliegt. Aber auch das Entwerfen solcher Filter durch den Webseitenbetreiber selbst ist denkbar. Diese Filterregeln sollten dann der Allgemeinheit über Listen zugänglich sein. Gerade für sicherheitsrelevante Anwendungen, wie Onlinebanking, kann dies eine zusätzliche Sicherheitsbarriere gegen Angriffe darstellen. Natürlich darf sich die Bank nicht darauf verlassen, dass die Daten, die am Webserver ankommen, keine Angriffe mehr enthalten. Denn einerseits kann ein Angreifer den Webserver direkt kontaktieren und andererseits kann ein findiger Hacker eine Möglichkeit gefunden haben, die Filter zu umgehen oder sie beim Opfer zu deaktivieren. Aber mit solchen Filtern kann eine Bank eine höhere Sicherheit für ihre Kunden erreichen und so das Vertrauen in Onlinebanking stärken.

Leider ist eine komplette Auflistung von empfohlenen Kriterien an dieser Stelle nicht sinnvoll. Einerseits werden die benötigten Funktionen bisher nur teilweise von aktuellen Browsern unterstützt, andererseits wäre hierfür eine längere, detailliertere Auswertung von vielen Webanwendungen notwendig, um eine Aussage treffen zu können. Es gibt zumeist nicht „den besten Filter“, sondern die Filterung ist immer davon abhängig, welche Funktionen ein Benutzer benötigt bzw. welche Webanwendungen er nutzt. Deshalb wird auch der Entwurf einer Filterliste, die möglichst viele Webanwendungen und Webseiten abdeckt, einige Zeit in Anspruch nehmen. Doch wäre die in solch einer Liste angelegte Zeit sinnvoll genutzt, wenn diese dann für die Mehrzahl der Anwender zweckmäßige Regeln enthält.

3.4.2 Request-Filter (URL-Filter und Form-Filter)

Die abgesendeten Daten müssen vor Missbrauch geschützt werden, deshalb dürfen sie Angreifern nicht in die Hände fallen. Da diese Daten aber auch auf anderem Wege ausgenutzt werden können, nämlich wenn ein Hacker ein Opfer oder dessen Browser dazu bringt, bestimmte, manipulierte Anfragen an einen Webserver zu stellen, muss auch der indirekte Missbrauch weitestgehend ausgeschlossen werden. Request-Filter geraten meist dort an ihre Grenzen, wo ein Angreifer es schafft, direkt manipulierte URLs oder Formulare, die der gleichen Domäne angehören, in eine Webseite einzuschmuggeln. Je nach vergebener Berechtigung kann aber auch hier noch ein Schutz gewährleistet werden, z. B. wenn eine manipulierte URL versucht, einem Applet Daten zu übermitteln. Die Request-Filterung wird in zwei Schritte eingeteilt. Zunächst wird eine Filterung durchgeführt, bevor eine Anfrage versendet wird. Danach wird die Antwort überprüft. Selbst wenn ein Request-Filter versagt, kann z. B. der Zugriff eines Applets auf die Webseite durch JavaScript-Filter blockiert werden.

3.4.2.1 Feststellen der Quelle der Anfrage

Zunächst muss der Browser die Quelle zuordnen, von der eine Anfrage kommt. So kann eine URL von Hand in die Adressleiste eingegeben, oder durch einen Klick im Browser angefordert worden sein. Aber auch ein anderes Programm kann URLs an den Browser übergeben. Je nachdem, woher der Aufruf stammt, stehen dem Browser verschiedene Informationen zur Verfügung. Leider übermittelt

das Betriebssystem nicht, von welchem Programm ein URL-Aufruf stammt. Deshalb kann der Browser bisher nur für angeklickte URLs oder abgesendete Formulare weitere Informationen ermitteln. Speziell bei Formularen können hierzu weitere Überprüfungen folgen. Könnte ein Browser auch das Programm ermitteln, von dem eine URL aufgerufen wurde, kann er weitere Einschränkungen vornehmen. So kann er z. B. festlegen, dass nur bestimmte Domänen von diesem aufrufbar sein sollten oder nur bestimmte MIME-Typen als Antwort erwartet werden. Es wäre auch möglich, dass bestimmte URLs nicht über die Adressleiste vom Browser aufgerufen werden dürfen. Dies scheint auf den ersten Blick wenig sinnvoll zu sein. Beim Firefox kann man allerdings beispielsweise über „about:config“ Einstellungen vornehmen. Gibt es nun in einem Mehrpersonenhaushalt Anwender, die gerne mal etwas ausprobieren aber kein technisches Verständnis besitzen, kann man so den Zugriff auf diese Einstellungsmöglichkeiten über diesen Weg unterbinden.

3.4.2.2 Überprüfen von Formular-Zielen

Um das Entführen oder Umleiten von Formulardaten zu vermeiden, muss der Browser sicherstellen, dass ein Formular an die URL gesendet wird, die der Benutzer mit diesem assoziiert. Deshalb sollte der Browser die Domäne des Ziels mit der Domäne der aufgerufenen URL vergleichen und eine Warnung ausgeben, falls diese nicht übereinstimmen. Wird das Formular in einem Frame angezeigt sollte(n) außerdem noch die Domäne(n) des Elternelements bzw. der Elternelemente abgeglichen werden. Nur wenn alle übereinstimmen, ist ein Angriff höchst unwahrscheinlich. Falls ein Formular diese Überprüfung nicht besteht, kann der Benutzer immer noch entscheiden, ob er das Formular absenden möchte oder nicht. Hierbei sollte der Browser die beschriebenen Regeln von [Schutz gegen Datenklau – Seite 40] beachten, um möglichst wenig unnötige Fehlermeldungen auszugeben, da diese den Anwender sonst abstumpfen lassen.

3.4.2.3 Automatisch versendete Formulare

Einige Angriffe beruhen auf automatisch abgesendeten Formularen. Eine Webanwendung sollte von sich aus erst einmal keine Formulare automatisch absenden können. In Zeiten von AJAX ist es aber wichtig, dass eine Webanwendung unter gewissen Umständen Formulare absenden kann. Allerdings sollte der Browser eine Differenzierung vornehmen, welche Webanwendungen Formulare abschicken können und welche Bedingungen hierfür nötig sind. Es wäre sinnvoll, hierfür mehrere Sicherheitsstufen für Webanwendungen vorzusehen.

Ich schlage folgende Sicherheitsstufen für den Formularversand vor (1 = größte Einschränkung, n+1 = nächst geringere Einschränkung nach n):

1. Keine automatisch versendeten Formulare zulassen. Event-Handler, wie onSubmit oder onClick, werden bei Formularen nicht beachtet.
2. Keine automatisch versendeten Formulare zulassen. Formulardaten dürfen nicht beim Absenden durch Event-Handler, wie onSubmit oder onClick, manipuliert werden.
3. Keinen automatischen Versand von Formularen zulassen, aber Manipulation der abgesendeten Daten durch Event-Handler erlauben.
4. Halbautomatischen Versand von Formularen an berechtigte Domänen akzeptieren. Mit einem halbautomatischen Versand ist gemeint dass ein weiteres Formular gesendet werden soll, z. B. durch JavaScript. Allerdings nur, wenn der Benutzer selbst aktiv eine Anfrage, z. B. durch Absenden eines Formulars oder klicken eines Links getätigt hat.

5. Automatischen Versand von Formularen an berechtigte Domänen bei Dateneingabe ermöglichen.
6. Vollautomatischen Versand von Formulardaten an berechtigte Domänen genehmigen.
7. Vollautomatischen Versand von Formulardaten an alle Domänen genehmigen.

Berechtigte Domänen sind hierbei - im Sinne von Kapitel [Schutz gegen Datenklau – Seite 40] - Domänen, mit denen die Quell-Domäne des Formulars kommunizieren darf. Außerdem beeinflussen die angegebenen Sicherheitsstufen keine anderen Filter, d. h. beispielsweise als „privat“ gekennzeichnete Daten sollten weiterhin nur in dafür berechtigte Formulare eingetragen werden dürfen. Für normale statische Webseiten ist die Sicherheitsstufe 3 empfehlenswert. Bei Webanwendungen mit dynamischen Verhalten sollte entweder die Stufe 4 oder 5 gewählt werden.

3.4.2.4 Verdächtige URLs

URLs, die ein „@“ (Klammeraffen) enthalten, sollten geblockt werden. Normalerweise dienen solche URLs zur Identifizierung eines Benutzers, allerdings werden diese auch immer häufiger zum Datenklau benutzt. Ein Anwender sieht die URL „www.example.com@2130706433“ und glaubt, dass diese zum Server „www.example.com“ führt. Dies ist allerdings nicht der Fall. Die Zeichenkette vor dem „@“ wird hierbei als Authentifizierungsparameter gedeutet, während die eigentliche URL nach dem „@“ folgt und hier als Ganzzahl angegeben wurde. Die Zahl „2130706433“ wird vom Browser als 127.0.0.1 interpretiert. Nur dem Browser selbst sollte es daher möglich sein, solche URLs anzufragen, indem er diese dynamisch aus Benutzername, Passwort und angefragter URL erstellt.

Außerdem sollten Parameter bei bestimmten URLs verworfen werden. So z. B. bei einem src-Attribut des IMG-Tag. Denn um ein Bild ausliefern zu können, benötigt der Webserver i. d. R. keine weiteren Parameter, deshalb sind auch solche URLs verdächtig. Allerdings kann es sein, dass ein Webserver bestimmte Bilder dynamisch erstellt und so Attribute wie „size=800x600“ unterstützt. Deshalb sollte es möglich sein, für diese Art von Einschränkungen Ausnahmen zu definieren. Außerdem kann man erwarten, dass eine URL zu einem Bild entweder keine Dateiendung trägt oder auf „.png“, „.gif“, „.jpeg“ bzw. „.jpg“ endet. Bei Applets könnte man im archive-Attribut „.jar“ als Endung erwarten und keine Endung oder „.class“ beim code-Attribut. Wenn gegen eine dieser Regeln verstoßen wird, sollte die URL nicht aufgerufen werden. Bei nicht akzeptierten Attributen könnte man diese auch weg lassen. Die möglichen Ausnahmeregelungen sollten immer kontextabhängig sein, also von der Domäne und dem verwendeten Tag bzw. Attribut.

Des Weiteren sollten Parameter nach bestimmten, potentiell gefährlichen Zeichen durchsucht werden. Hierbei sollte zuerst die Codierung der URL ermittelt werden. Ist diese nicht ermittelbar oder enthält die URL Zeichen, die nicht in URLs vorkommen dürfen, wie die meisten Sonderzeichen, Umlaute oder Steuerzeichen, so sollte die URL nicht aufrufbar sein. Falls ein Parameter Zeichen enthält, die vom Zielsystem interpretiert werden könnten, wie HTML-spezifische Zeichen, so sollte der Browser zumindest eine Warnung ausgeben, bevor er die Anfrage absendet.

3.4.2.5 Antwort überprüfen

Der Browser sollte immer den MIME-Typ der Antwort überprüfen und mit einer Liste von erwarteten MIME-Typen abgleichen. Erwartet der Browser z. B. ein JAR-Archiv und erhält den MIME-Typ „image/gif“ zurück, so sollte er diese Antwort nicht weiter verarbeiten, sondern verwerfen und so weiter machen, als wäre der Inhalt nicht verfügbar. Der Browser sollte zunächst immer den MIME-Typ überprüfen, bevor er die Verarbeitung durch externe Plugins, wie der JVM, zulässt. Würden die Browser dies berücksichtigen, wäre eine GIFAR-Attacke nicht möglich.

3.4.2.6 Spezialfilter

Die größte Sicherheit kann auch bei dieser Art von Filtern dadurch erreicht werden, dass diese optimal auf die Webanwendung abgestimmt werden. Dies kann durch Profis vorgenommen werden. Allerdings ist es für Außenstehende, auch wenn es technisch versierte Leute sind, schwer, solche Filter auf eine Webanwendung abzustimmen, da sie eine Menge an Quelltext durchforsten oder per try-and-error-Verfahren alle möglichen Funktionen testen müssten. Deshalb ist es empfehlenswert, dass die Request-Filter von den Anwendungsentwicklern selbst erstellt werden, da diese genau wissen, welche Anfragen und Antworten ihre Anwendung versendet.

4 Schutzmaßnahmen der Browser

Dieses Kapitel befasst sich mit den Schutzmaßnahmen der zwei beliebtesten Browser Microsoft Internet Explorer 7 und Mozilla Firefox 3. Zunächst lege ich dar, wie sich die Browser mit Standardeinstellungen verhalten. Anschließend gebe ich eine Empfehlung und Erklärung über die Einstellungsmöglichkeiten ab, die die Sicherheit für den Anwender verbessern.

4.1 Testmethodik

Für die Browsertest verwende ich zumeist eigens erstellte Testseiten. In manchen Fällen nutze ich bereits vorhandene Testseiten im Internet, auf diese weise ich dann hin. Folgende Tests werden mit den Browsern durchgeführt:

4.1.1 Java-Applet-Test

Es wird versucht eine Seite mit einem Java-Applet zu laden,

- bei der das Applet eine einzelne Klasse darstellt und
  - sich diese Klasse innerhalb derselben Domäne wie die Webseite befindet.
  - sich diese Klasse innerhalb einer anderen Domäne befindet.
- bei der sich das Applet in einem JAR-Archiv befindet und
  - sich dieses Archiv innerhalb derselben Domäne wie die Webseite befindet.
  - sich dieses Archiv innerhalb einer anderen Domäne befindet.

Hierbei soll überprüft werden, ob der Browser automatisch Java-Applets ausführt. Da Applets zu den aktiven Inhalten gehören, sind sie potentiell schädlich. Außerdem wird überprüft, ob der Browser dabei Unterschiede zwischen Applets macht, die aus einer Klasse bestehen, und solchen, die sich in einem JAR-Archiv befinden.

Führt ein Browser Applets von derselben Domäne aus, in der sich die Webseite befindet, so können nur in diese Domäne eingeschleuste Applets Schaden anrichten. Allerdings ist es einem Angreifer zumeist nicht möglich, ein Applet in eine Domäne einzuschleusen, falls es auch normalen Benutzern nicht regulär möglich ist, dies zu tun. Eine Ausnahme hierfür sind durch GIFAR-Attacken eingeschleuste Applets. Allerdings könnten diese noch durch Überprüfung des MIME-Typs enttarnt werden.

Akzeptiert ein Browser allerdings auch Applets anderer Seiten, so können diese z. B. Daten klauen oder das Opfer auf eine andere Seite umleiten. Solche Applets können beispielsweise in von Nutzern gestaltete Seiten oder aber von Angreifern durch XSS-Attacken eingebunden werden. Ein Angreifer könnte dies auch nutzen, um Applets von Seiten, die er angreifen will, in seine Seite zu integrieren. Wenn sich diese Applets über JavaScript steuern lassen, kann ein Hacker somit Methoden des Applets im Namen des Opfers aufrufen, falls dieser JavaScript erlaubt.

4.1.2 MIME-Typ-Überprüfung bei Archiven für Applets

Die Testseite verwendet eine GIFAR-Datei als Archiv in einem Applet-Tag. Die Datei befindet sich zudem in einer anderen Domäne als die Webseite. Dieser Test ähnelt sehr dem, mit einem normalen JAR-Archiv, das aus einer anderen Domäne stammt. Der Unterschied hierbei liegt darin, dass der Webserver als MIME-Typ „image/gif“ ausliefert. Es soll überprüft werden, ob der Browser das Applet trotz des falschen MIME-Typs ausführt.

Falls das Applet trotz eines falschen MIME-Typs vom Browser akzeptiert wird, ist der Weg für GIFAR-Attacken frei.

4.1.3 Cookie-Test für Applets

Bei diesem Test wird ein Applet verwendet, das versucht die Cookies auszulesen und vorhandene Cookies bei einem URL-Aufruf mitzuverwenden (Gagnon, 1997).

Hierbei wird überprüft, ob Cookies für Applets mit Hilfe von JavaScript les- und schreibbar sind bzw. ob Cookies bei Anfragen des Applets mit zum Webserver gesandt werden. Für den JavaScript-Ansatz kommen hierbei zwei verschiedene Varianten zum Einsatz. Zum einen wird die Eval-Methode mit „document.cookie“ aufgerufen und zum anderen wird versucht auf die Cookies über „getMember("cookie")“ zuzugreifen (Sun Microsystems, Inc., 2008).

Wenn sich Cookies über JavaScript auslesen oder setzen lassen, ist es möglich die Session durch eingeschleuste Applets zu stehlen. Außerdem können dann auch weitere JavaScript-Aktionen von solchen Applets ausgeführt werden. Da die Eval-Methode sehr mächtig ist und JavaScript auch Zugriff auf das DOM besitzt, kann durch ein Applet die beim Benutzer angezeigte Seite somit fast beliebig verändert werden.

Kann ein Applet bestehende Cookies bei URL-Aufrufen mit verwenden, so sind CSRF-Attacken möglich (Sun Microsystems, Inc., 2008). Diese können einerseits durch eingeschleuste Applets durchgeführt werden. Falls sich bestehende Applets einer Webanwendung über JavaScript steuern lassen, kann ein Angreifer diese andererseits auch einfach auf seiner eigenen Webseite einbinden und dann über Methoden-Aufrufe CSRF-Attacken durchführen.

4.1.4 JavaScript-Tests

Des Weiteren werden durch mehrere Webseiten verschiedenste JavaScript-Funktionalitäten getestet. Da JavaScript eine Vielzahl von Objekten, Methoden und Attributen besitzt, wäre ein Test aller dieser sehr langwierig. Deshalb werden die Tests eingeschränkt. Nachfolgend ein paar Beispiele:

- Aufruf der Alert-Methode.
- Setzen von HTML-Element-Werten mit Hilfe von „document.getElementById()“ und „*htmlElement*.value="Wert"“.
- Lesen von Objekten bzw. Attributen, wie document.cookie, document.URL, location, usw.
- Schreiben in das HTML-Dokument mit document.write.

Die Tests sollen zeigen, ob der Browser standardmäßig JavaScript ausführt. Falls er dies tut, wird überprüft, ob er den Zugriff auf gewisse Objekte, Attribute oder Methoden verweigert. Diese Einschränkungen werden allerdings den Sicherheitseinstellungen, sowie Dokumentationen entnommen, weil vollkommen umfassende Tests von JavaScript nicht durchgeführt werden.

Da JavaScript zu den aktiven Inhalten gehört, kann es auch Schaden anrichten. Heutzutage kommen viele Webanwendungen allerdings nicht ohne JavaScript aus. Können Benutzer bei Anwendungen selbst Inhalte bestimmen und sogar JavaScript einbinden, kann dies gefährlich für andere werden. Außerdem wird für die meisten XSS-Attacken JavaScript-Unterstützung benötigt. Aber auch bei anderen Angriffen, wie Clickjacking wird auf JavaScript gesetzt. Nimmt der Browser also keine Einschränkungen vor, so ist es für Hacker leichter, Angriffe durchzuführen.

4.1.5 URL-Überprüfung

Dieser Test soll zeigen, ob ein Browser potentiell gefährliche Zeichenketten in URLs erkennt. Hierzu werden die in Abschnitt [Cross-Site-Scripting (XSS) – Seite 10] beschriebenen Angriffe durchgeführt. Erkennt der Browser gefährliche URLs, kann er XSS-Attacken ggf. verhindern.

4.1.6 Frame-Bursting-Test

Frame-Bursting nennt man das Zerstören eines Frames. Damit ist gemeint, dass eine Webseite, die innerhalb einer anderen in einem Frame angezeigt werden soll, dies erkennt und dies unterbindet, indem sie für den Top-Frame bzw. das Fenster die URL abändert, so dass die Seite neu geladen wird. Frame-Bursting verhindert somit, dass andere Webpräsenzen den Seiteninhalt, der durch diese Technik geschützten Webseite, einbinden. Allerdings benötigt Frame-Bursting JavaScript.

Frame-Bursting wird jeweils einmal innerhalb eines Framesets und einmal innerhalb eines IFrames getestet.

Wenn der Browser Frame-Bursting verhindert, können Seiteninhalte auch in fremden Seiten eingebaut werden. Somit können unbedarfte Benutzer irregeführt werden. Außerdem bauen verschiedene Angriffe darauf, Webseiten in Frames anzuzeigen, beispielsweise Clickjacking.

4.1.7 Frame-Test

Es wird wie im vorher erwähnten Testszenario überprüft, wie ein Browser beim Einbinden einer Webseite in einem Frame reagiert. Allerdings sind diesmal die eingebunden Seiten nicht durch Frame-Bursting geschützt.

Wie auch im Frame-Bursting-Test werden die Seiten einmal in einem Frameset und das andere mal in einem IFrame geladen. Jedoch wird der Test jeweils mit einer Seite aus der gleichen und danach mit einer Seite aus einer anderen Domäne durchgeführt.

Der Test soll zeigen, ob ein Browser Webseiten ohne Warnung in Frames einbindet. Während dies bei Webseiten aus der gleichen Domäne meist harmlos ist, kann es bei Webseiten, die aus verschiedenen Domänen stammen, gefährlich sein. Clickjacking-Angriffe erfolgen normalerweise aus einer anderen Domäne heraus. Würde der Browser Frames aus fremden Domänen nicht automatisch laden, wäre Clickjacking also nicht oder nur schwer durchführbar.

4.1.8 Klicken auf unsichtbare Elemente

Um zu überprüfen, ob der Browser den Anwender warnt, wenn dieser auf unsichtbare Elemente klickt, werden Clickjacking-Demo-Seiten verwendet. Hierbei sollte der Browser erkennen, dass der Benutzer getäuscht wird und geeignete Maßnahmen ergreifen. Erkennt der Browser das Täuschungsmanöver nicht, so kann ein Angreifer Clickjacking-Attacken verwenden.

Die verwendeten Seiten sind:

- http://guya.net/security/clickjacking/game.html (Aharonovsky, 2008)
- http://grack.com/record/ (Mastracci, 2008)

4.1.9 Laden von Links anderer Domänen

Bei diesem Test wird überprüft, wie der Browser mit URLs umgeht, die auf andere Domänen verweisen. Speziell solche, die automatisch geladen werden, wie URLs des src-Attributes im IMG-Tag, können für Angriffe benutzt werden.

Es wird hierzu einmal eine URL im src-Attribut des Bildes angegeben, die vom Webserver erzeugt wird und URL-Parameter enthält. Der zweite IMG-Tag und damit auch die URL wird dynamisch per JavaScript mit Hilfe von document.write() generiert und liest auch das Cookie der angezeigten Seite mit aus.

Der Test soll zeigen, ob URLs aus anderen Domänen automatisch aufgerufen werden. Falls dies der Fall ist, wird darauf geachtet, ob der Browser bestimmte Einschränkungen vornimmt, z. B. Cookies nicht mit sendet oder URL-Parameter weg lässt. Solche automatisch aufgerufenen URLs werden besonders gerne für CSRF-Attacken verwendet. Nimmt der Browser keine Einschränkungen vor, ist der Benutzer also nicht vor solchen Angriffen geschützt.

4.2 Internet Explorer

Der Internet Explorer nutzt ein Zonenmodell. Jede Webseite wird einer Zone zugeteilt. Der Browser bietet die Sicherheitszonen „Internet“, „Lokales Intranet“, „vertrauenswürdige Sites“ und „eingeschränkte Sites“ an. Eine Seite ist standardmäßig der Zone „Internet“ zugeordnet. Die Zone „Lokales Intranet“ ist für Seiten in Firmennetzen gedacht. Besonders vertrauenswürdige Seiten gehören in die gleichnamige Zone, nicht vertrauenswürdige in die eingeschränkte. Leider müssen die Seiten von Hand eingepflegt werden. In einer administrierten Umgebung ist dies oft der Fall. Für Privatnutzer ist der Pflegeaufwand meist zu hoch oder das technische Wissen nicht vorhanden. Die Zonen lassen sich auf unterschiedliche Sicherheitsstufen stellen. In der Zone „Internet“ können die Stufen „mittel“, „mittelhoch“ und „hoch“ gewählt werden. Bei der Zone für eingeschränkte Webseiten ist die Stufe „hoch“ fest eingestellt. Für die beiden anderen Zonen kann jeweils eine von fünf Stufen zwischen „sehr niedrig“ und „hoch“ gewählt werden. Da das Intranet normalerweise sehr vertrauenswürdig ist, ist die Standardeinstellung hierfür „niedrig“. Bei vertrauenswürdigen Seiten ist „mittel“ voreingestellt. Das Internet wird standardmäßig auf die Stufe „mittelhoch“ gesetzt. Die Sicherheitseinstellungen sind je nach verwendeter Sicherheitsstufe unterschiedlich festgelegt.

4.2.1 IE mit Standardeinstellungen

Zunächst habe ich den Internet Explorer auf Standardeinstellungen zurückgesetzt und keine weiteren Einstellungen vorgenommen. Somit werden für alle Seiten die Sicherheitsregeln der Zone „Internet“ verwendet. Allerdings sind bei mir einige Add-Ons, wie Flash von Adobe, Java von Sun, Silverlight von Microsoft usw. installiert.

Die Einstellungen des Internet Explorers wurden mit den beschriebenen Methoden getestet. Die Ergebnisse der Tests sind nachfolgend aufgeführt.

Negative Ergebnisse:

- Der Browser akzeptiert Applets ohne Nachfrage, auch wenn diese von anderen Domänen stammen. Er macht dabei keine Unterschiede zwischen einzelnen Klassen und JAR-Archiv.
- Der MIME-Typ von Archiven für Applets wird nicht überprüft.
- Cookies sind für Applets per JavaScript les- und schreibbar. Applets können bestehende Cookies für eigene Verbindungen nutzen.
- JavaScript wird fast ohne Einschränkungen ausgeführt. Folgende Beschränkungen liegen vor:
    - Die Größe und Position von Fenstern muss bestimmte Kriterien erfüllen.
    - Die Statusleiste darf nicht abgeändert werden.
    - Eingabeaufforderungsfenster von Skripts werden unterbunden.
- Adressen werden nicht auf gefährlichen Code hin überprüft.

- In Frames können Seiten einer anderen Domäne ohne Warnung eingebunden werden, egal ob es sich um einen IFrame oder ein Frameset handelt.
- Ein Benutzer wird nicht gewarnt, wenn er unsichtbare Elemente anklickt.
- Links, die auf andere Domänen verweisen, beispielsweise in IMG-Tags, werden nicht gesondert behandelt und ggf. automatisch geladen. Auch die bereits vorhandenen Cookies stehen ihnen zur Verfügung.

Positive Ergebnisse:

- Frame-Bursting funktioniert sowohl bei Framesets als auch bei IFrames.

4.2.2 Einstellungsmöglichkeiten beim IE

Der Internet Explorer ist für fast alle Angriffe auf Webanwendungen bzw. Benutzer offen. Lediglich das Ausführen von Frame-Bursting-Code kann einige Angriffe abwehren, wie Clickjacking. Allerdings muss hierzu die Webanwendung, auf die es ein Angreifer abgesehen hat, auch Frame-Bursting-Code verwenden, um einen solchen abzuwehren. Es ist also noch eine Menge Verbesserungspotential vorhanden. Nun sollen die Einstellungsmöglichkeiten beleuchtet werden, die eine höhere Sicherheit im IE bieten. Allerdings gibt es meist nicht die perfekten Einstellungen. Deaktiviert man z. B. Skripting im Internet Explorer, so funktioniert auch Frame-Bursting nicht mehr. Außerdem benötigen manche Webseiten einige Features, die den Browser unsicherer machen, wenn man nicht auf Komfort verzichten möchte. Um einen Überblick über die Einstellungsmöglichkeiten bei IE zu Erhalten, werden im Folgenden die Erklärungen von (Born, 2004; Heise Security, 2006; Janowicz, 2006; Nasarek, 2007; Netzwelt, 2005; Zocholl, 2005) herangezogen.

Die Einstellungen sind im Internet Explorer über Extras → Internetoptionen erreichbar. Der größte Teil der sicherheitsrelevanten Einstellungen befindet sich im Reiter „Sicherheit“.

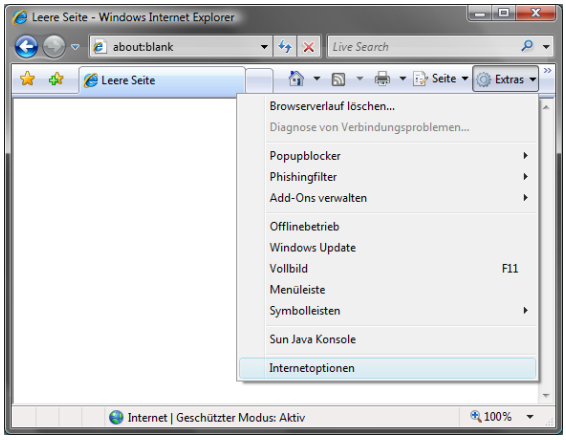

Abbildung 4-1: IE: Extras – Internetoptionen

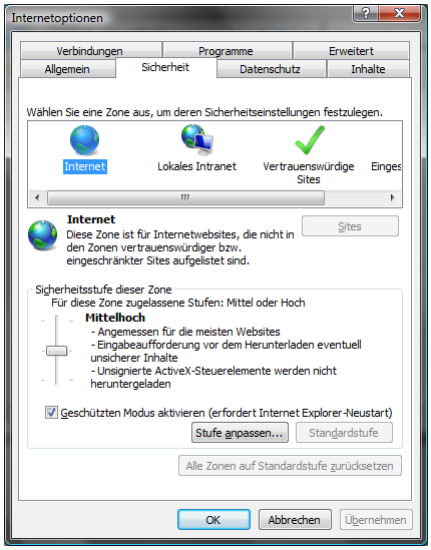

Abbildung 4-2: IE: Internetoptionen – Sicherheit

Die Einstellungen sind für verschiedene Sicherheitszonen unterschiedlich einstellbar. Dies kann mit Hilfe eines Schiebereglers stufenweise geschehen. Der Benutzer kann zudem noch einzelne Einstellungen abändern indem er auf „Stufe anpassen …“ klickt.

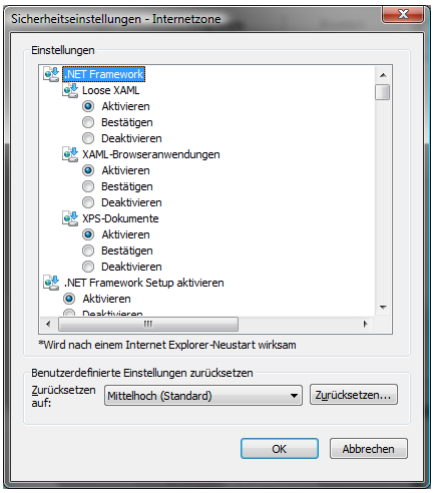

Abbildung 4-3: IE: Sicherheitsstufe anpassen

Daraufhin bekommt er die einzelnen Einstellungsmöglichkeiten angezeigt und kann diese nach Bedarf anpassen. Neben diesen Einstellungen gibt es noch weitere, die für die Daten- und Anwendungssicherheit verantwortlich sind. Dies sind zum einen die Einstellungen für Autovervollständigung im Reiter „Inhalte“. Zum anderen ist dies die Add-On-Verwaltung im Reiter „Programme“. Außerdem gibt es noch einige sicherheitsrelevante Einstellungen im Reiter „Erweitert“.

Leider findet ein Anwender weder über die Hilfe noch auf der Internet-Präsenz von Microsoft brauchbare Hinweise oder Erklärungen der einzelnen Einstellungen. Dies ist sehr schlecht, da ein Benutzer keine Möglichkeit hat, sich selbst ein Bild von der Güte der Sicherheitseinstellungen zu schaffen. Noch gefährlicher ist es, wenn Benutzer mit Halbwissen meinen, eine Einstellung würde etwas ganz anderes bewirken. Dies ist sogar teilweise in Fachliteratur zu erkennen, in welcher Einstellungen, wie „Subframes zwischen verschiedenen Domänen bewegen“, falsch beschrieben sind. Dies kann gefährliche Auswirkungen haben.

4.2.2.1 Zoneneinstellungen

Zunächst werde ich die Sicherheitseinstellungen erläutern, die zu sehen sind, wenn man im Reiter Sicherheit der Internetoptionen auf „Stufe anpassen …“ klickt:

4.2.2.1.1 .NET

Einige Benutzer haben in Verbindung mit dem Internet schon einmal den Begriff „.NET“ gehört und denken deshalb wohl, dass auch .NET im Browser aktiviert werden muss um auf einigen Seiten alle Funktionen nutzen zu können. .NET-Anwendungen laufen jedoch normalerweise auf dem Webserver. Deshalb können alle .NET-Funktionen deaktiviert werden. Hierzu gehören die Bereiche „.NET Framework“ mit „Loose XAML“, „XAML Browseranwendungen“ und „XPS Dokumente“, sowie „.NET Framework Setup“. Außerdem gibt es noch einen Bereich „Auf .NET Framework basierende Komponenten“, bei dem „Ausführen von Komponenten, die mit Authenticode signiert sind“, „Ausführen von Komponenten, die nicht mit Authenticode signiert sind“ und „Berechtigungen für Komponenten mit Manifesten“ deaktiviert werden sollten.

4.2.2.1.2 ActiveX-Steuerelemente und Plugins

ActiveX-Komponenten sind eine große Sicherheitsbedrohung beim Internet Explorer. Viele Angriffe auf den IE nutzen diese Komponenten aus. Deshalb sollte die Nutzung dieser möglichst eingeschränkt werden. Unsignierte bzw. unsichere ActiveX-Komponenten sollte man keinesfalls benutzen, da nicht sichergestellt werden kann, von welchem Anbieter diese Komponenten stammen bzw. ob sie manipuliert wurden. Darum sollten die Einstellungen „ActiveX-Steuerelemente initialisieren und ausführen, die nicht als "sicher für Skripting" markiert sind“ und „Unsignierte ActiveX-Steuerelemente herunterladen“ deaktiviert werden. Außerdem sollte man bestimmte Einstellungen, die automatische Aktionen triggern, deaktivieren. Dies ist zum einen „Ausführen von bisher nicht

verwendeten ActiveX-Steuerelementen ohne Eingabeaufforderung zulassen“. Somit bekommt man eine Meldung, wenn ein bisher noch nicht verwendetes ActiveX-Steuerelement angesprochen wird.

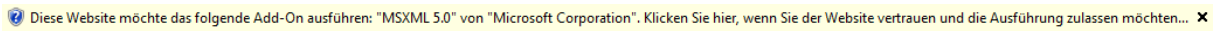

Abbildung 4-4: IE: Meldung – Noch nicht verwendetes ActiveX-Steuerelement ausführen

Zum anderen sollte die Aktion „Automatische Eingabeaufforderung für ActiveX-Steuerelemente“ nicht zugelassen werden. Wird diese jedoch aktiviert, erhält man keine Meldung, wenn ein neues ActiveX-Steuerelement installiert werden soll, obwohl die Einstellung „Signierte ActiveX-Steuerelemente herunterladen“ bzw. „Unsignierte ActiveX-Steuerelemente herunterladen“ auf „Bestätigen“ gesetzt wurde.

Des Weiteren sind die Einstellungen „Binär- und Skriptverhalten“, „Skriptlets zulassen“ und „Videos und Animationen auf einer Webseite anzeigen, die keine externe Medienwiedergabe verwenden“ als unsicher einzustufen und veraltet oder werden nur selten verwendet. Aus diesen Gründen sollte man diese auch abschalten.

Falls man dennoch auf ActiveX-Steuerelemente und Plugins nicht verzichten kann, sind die im Folgenden erläuterten Einstellungen interessant:

Durch die Einstellung „ActiveX-Steuerelemente ausführen, die für Skripting sicher sind“ kann der Benutzer über die Aktivität als sicher geltender ActiveX-Steuerelemente bestimmen, die von Skripts aus der Seite heraus angesprochen werden sollen. Bei einigen Webseiten, wie YouTube, ist solch ein Zugriff nötig, andere funktionieren auch ohne diesen ohne merkliche Änderungen. Wenn ActiveX-Steuerelemente und Plugins nicht benötigt werden, sollte man diese Funktion deaktivieren. Benötigt man den Zugriff nicht oft, ist es ratsam die Option „Bestätigen“ zu wählen. So erhält man immer eine Eingabeaufforderung zum Zulassen oder Verweigern dieser Einstellung. Allerdings können diese ständigen Meldungen, die durchaus auch mehr als einmal pro Seite erscheinen, sehr lästig sein. Wenn man die Option „ActiveX-Steuerelemente und Plugins ausführen“ nicht auf aktiviert stellt, ist der zusätzliche Sicherheitsgewinn der Einstellung „ActiveX-Steuerelemente ausführen, die für Skripting sicher sind“ nicht sehr groß. Aus diesem Grunde kann man dann für diese Einstellung auch „Aktivieren“ wählen. Wenn man noch mehr auf Sicherheit achten möchte und einen Meldungen nicht zu sehr nerven, kann man auch „Bestätigen“ einstellen. Wird die Einstellung deaktiviert, werden einige Funktionen auf Webseiten nicht mehr arbeiten. Dies ist also nur empfehlenswert, wenn man keine Plugins nutzen möchte oder die Seiten, die diese benötigen in die Zone „Vertrauenswürdige Sites“ einpflegt und dort die Einstellungen entsprechend lockert.

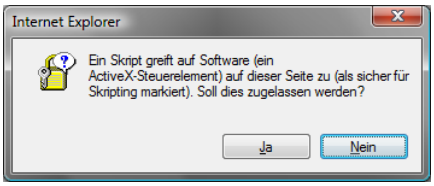

Abbildung 4-5: IE: Meldung – Skriptzugriff auf ActiveX-Steuerelement

Die Einstellung „ActiveX-Steuerelemente und Plugins ausführen“ gehört zu den wichtigsten bezüglich der Sicherheit im Internet. Sie bestimmt, was geschehen soll, wenn eine Webseite ein ActiveX-Steuerelement bzw. ein Plugin ausführen möchte. Wenn diese Inhalte für den Benutzer nicht von Bedeutung sind, sollte man die Option „Deaktivieren“ wählen. Dies bietet die höchste Sicherheit.

Werden nur wenige Seiten benötigt, die solche Elemente verwenden, kann man diese in die Zone der vertrauenswürdigen Seiten eintragen und dort ActiveX-Steuerelemente und Plugins erlauben oder deren Ausführung bestätigen lassen. Allerdings wundert man sich bei manchen Seiten vielleicht, wenn etwas nicht funktioniert, da der Internet Explorer nicht immer eine Informationsmeldung bringt, dass die Seite ein Plugin oder ActiveX-Steuerelement verwenden möchte. Bei einigen Seiten erscheint aber auch eine Informationsmeldung.

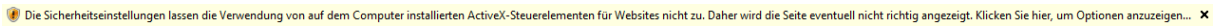

Abbildung 4-6: IE: Meldung – Sicherheitseinstellungen blockieren ActiveX-Steuerelement

Wenn man wissen möchte, ob eine Webseite ActiveX-Steuerelemente oder Plugins benutzt, sollte man die Option „Bestätigen“ auswählen. Es ist nicht empfehlenswert, die Einstellung auf „Aktivieren“ zu setzen, da Plugins und ActiveX-Steuerelemente zu den aktiven Inhalten zählen und deshalb gefährlich sein können. Ob man die Einstellung deaktivieren sollte oder sich besser Bestätigungsmeldungen anzeigen lässt, hängt von den Bedürfnissen des jeweiligen Benutzers ab. Die größtmögliche Sicherheit bringt das Deaktivieren für die Internetzone und die Option „Bestätigen“ für die Zone der vertrauenswürdigen Seiten. Wenn man die Sicherheitsmeldungen bei vertrauenswürdigen Seiten sowieso bestätigt, ist auch die Option „Aktivieren“ für die vertrauenswürdige Zone sinnvoll. Will man immer aktuell entscheiden, ob Plugins oder ActiveX-Steuerelemente verwendet werden dürfen, sollte für die Internet-Zone „Betätigen“ gewählt werden. Dann muss man aber auch mit vielen Meldungen während des Surfens rechnen. Ist diese Einstellung deaktiviert, sind alle anderen für ActiveX hinfällig, da überhaupt keine ActiveX-Steuerelemente geladen oder installiert werden.

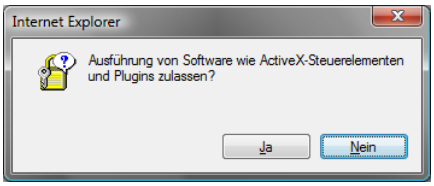

Abbildung 4-7: IE: Meldung – ActiveX-Steuerelemente und Plugins zulassen

Wenn eine Webseite ein ActiveX-Steuerelement verwendet, das noch nicht installiert ist, kann der Internet Explorer dies automatisch tun. Ob er das auch soll, kann mit „Signierte ActiveX-Steuerelemente herunterladen“ festgelegt werden. Allerdings gilt dieses Verhalten nur für signierte ActiveX-Steuerelemente. Die entsprechende Einstellung für nicht signierte ActiveX-Steuerelemente, nämlich „Unsignierte ActiveX-Steuerelemente herunterladen“, sollte auf jeden Fall deaktiviert sein, wie weiter oben bereits erwähnt. Für signierte ActiveX-Steuerelemente kann man dies mit ruhigem Gewissen auf „Bestätigen“ stellen. Somit erhält man immer eine Meldung, wenn ein neues ActiveX-Steuerelement heruntergeladen werden soll, außer die Einstellung „Automatische Eingabeaufforderung für ActiveX-Steuerelemente“ wurde aktiviert, was nicht empfehlenswert ist.

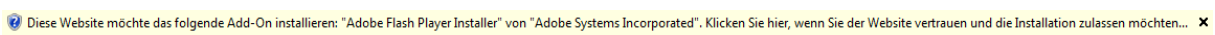

Abbildung 4-8: IE: Meldung – ActiveX-Steuerelement installieren

Die Entscheidung, ob das ActiveX-Steuerelement wirklich installiert werden soll, sollte der Benutzer selbst treffen. Wenn man seine ActiveX-Steuerelemente nicht über den Browser installiert, sondern z. B. durch Offline-Installationsdateien, kann diese Einstellung auch deaktiviert werden. Sie auf

„Aktivieren“ zu setzen ist nicht empfehlenswert, da sonst signierte ActiveX-Steuerelemente automatisch installiert werden können.

4.2.2.1.3 Benutzerauthentifizierung

Beim Bereich „Benutzerauthentifizierung“ sollte „Nach Benutzername und Kennwort fragen“ gewählt werden. So bekommt der Anwender immer mit, wenn eine Authentifizierung erforderlich ist.

4.2.2.1.4 Download

Da man in der Regel nicht auf Dateidownloads verzichten möchte, sollte man im Abschnitt „Download“ den „Dateidownload“ aktivieren. Die „Automatische Eingabeaufforderung für Dateidownloads“ sollte deaktiviert werden, so dass nur Dateien heruntergeladen werden, wenn man dies selbst veranlasst. Will eine Seite einen automatischen Download starten, erhält man dann eine Meldung.

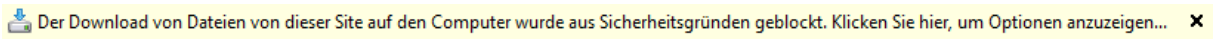

Abbildung 4-9: IE: Meldung – Automatischer Download geblockt

Den „Schriftartdownload“ sollte man auf „Bestätigen“ stellen, wenn man auch außergewöhnliche Schriftarten auf Webseiten nutzen möchte. Ist dies nicht der Fall kann man diese Funktion auch deaktivieren.

4.2.2.1.5 Skripting

Ein weiterer wichtiger Bereich ist das „Skripting“. Hier sollten nach Möglichkeit alle Optionen deaktiviert werden. Da man aber sicher auch Webseiten besucht, für die man Skripting aktiveren möchte, sollte man diese dann in die Zone „Vertrauenswürdige Sites“ einfügen und dort die Einstellungen soweit lockern, wie dies nötig ist. Allerdings sollte man die Einstellungen „Eingabeaufforderung für Informationen mithilfe von Skriptfenstern für Websites zulassen“, „Programmatischer Zugriff auf die Zwischenablage zulassen“ und „Statuszeilenaktualisierung über Skript zulassen“ auf jeden Fall deaktivieren. Die Gründe dafür sind folgende: Manche Anwender halten Skriptfenster für Eingabeaufforderungen des Browsers und geben dort deshalb ein Passwort oder andere Informationen ein, die missbraucht werden können. Zugriffe auf die Zwischenablage können für Anwender gefährlich sein, da sich dort alle möglichen Informationen befinden können. Je nachdem was der Benutzer zuletzt kopiert hat, kann dies beispielsweise ein Text mit privaten Angaben oder eine Datei mit E-Mailadressen usw. sein. Außerdem lassen sich Webanwendungen auch ohne Zugriff auf die Zwischenablage entwerfen. Eine Anwendung die einen solchen benötigt, erscheint also verdächtig. Änderungen in der Statusleiste können den Anwender täuschen. Deshalb kann es gefährlich sein, wenn diese für Skripte ermöglicht werden. Setzt man den Zugriff auf die Zwischenablage auf „Bestätigen“ kommt folgende Meldung:

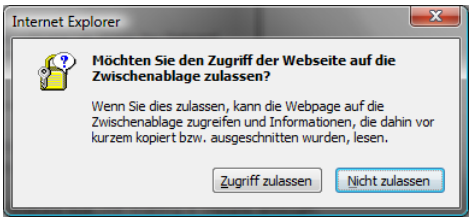

Abbildung 4-10: IE: Meldung – Zugriff auf Zwischenablage zulassen

Mit „Active Scripting“ kann man Skripts zulassen. Deaktiviert man diese Einstellung, so können keine Skripts ausgeführt werden. Somit werden viele Angriffe undurchführbar, vor allem solche die Daten klauen. Allerdings sind heutzutage die meisten Seiten auf Skripte angewiesen. Deshalb wird für viele Benutzer eine Deaktivierung für die Internetzone und nur die Aktivierung für die vertrauenswürdige Zone, wie oben erwähnt, nicht in Frage kommen. Die Option „Bestätigen“ zu wählen ist kaum sinnvoll, da der Anwender dadurch mit einer Vielzahl von Meldungen überflutet wird. Somit wird es schwer sein, die richtigen Entscheidungen zu treffen. Auf die Dauer stumpft man deshalb ab und liest die Meldungen gar nicht mehr durch. So klickt man vielleicht auch ähnlich aussehende Meldungen weg, die eine andere Bedeutung haben, da man den Text gar nicht mehr durchliest. In einem solchen Fall ist es besser die Option „Aktivieren“ zu wählen. Allerdings muss man sich bewusst sein, dass dies eine eklatante Verschlechterung der Sicherheit bedeutet.

Über „Skripting von Java-Applets“ kann man bestimmen, ob Skripts Methoden von Applets aufrufen können. Wird dies zugelassen, können Daten zwischen dem Skript und dem Applet ausgetauscht werden. Dies kann problematisch sein, da sich die Domänen von Skript und Applet unterscheiden können und so die Daten die Domänengrenzen überschreiten können, ohne dass der Browser Einschränkungen vornimmt. Da diese Funktionalität allerdings nicht oft von Webseiten benutzt wird, ist es sinnvoll die Option „Bestätigen“ für diese Einstellung zu wählen. So kann man jeweils entscheiden, ob die Skripts der Seite auf ein Applet zugreifen dürfen oder nicht. Deaktiviert man die Einstellung, ist man auf der sicheren Seite, muss aber ggf. auf Funktionen einer Webapplikation verzichten. Wenn man „Aktivieren“ auswählt, können auch gefährliche Webseiten Applets einbinden und Methoden dieser aufrufen. Sind dies Applets, die zu einer anderen Webapplikation gehören und sogar eine authentifizierte Verbindung mitbenutzen, so kann ein Angreifer hierdurch die andere Anwendung manipulieren. Deshalb ist es nicht ratsam „Skripting von Java-Applets“ zu aktivieren.

4.2.2.1.6 Verschiedenes

Im letzten Bereich „Verschiedenes“ befinden sich noch weitere Einstellungsmöglichkeiten. Mit „Anwendungen und unsichere Dateien starten“ kann man bestimmen, ob unsichere Dateien, wie *.exe oder *.vb über den Internet Explorer direkt ausgeführt werden dürfen. Dies stellt eine sehr große Gefahr da. Deaktivieren bietet zwar den besten Schutz, allerdings können dann auch solche Dateien nicht mehr heruntergeladen werden. Deshalb sollte man hier „Bestätigen“ wählen, wenn man nicht ganz auf solche Dateien verzichten kann.

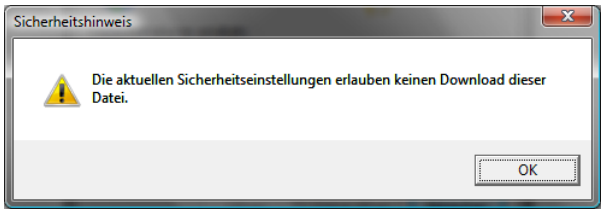

Abbildung 4-11: IE: Meldung – Dateidownload verweigert

Durch die Einstellung „Auf Datenquellen über Domänengrenzen hinweg zugreifen“ gibt man an, ob Komponenten domänenübergreifende Verbindungen zu Datenquellen herstellen dürfen. Solche Zugriffe sind sehr kritisch zu betrachten, deshalb sollte man diese Einstellung deaktivieren. Mit „Dateien basierend auf dem Inhalt und nicht der Dateierweiterung öffnen“ kann man bestimmen, dass die MIME-Typ-Überprüfung umgangen wird. Der Internet Explorer versucht dann anhand von Bitfolgen den Dateityp zu bestimmen und nicht anhand des MIME-Typs. So können auch Dateien mit falschem MIME-Typ dargestellt werden. Allerdings sollte ein Webserver wissen, welche Dateien er

ausliefert. Wird ein falscher MIME-Typ gesendet, kann ein Angreifer auch versucht haben, eine Datei zu tarnen. Deshalb sollte diese Funktion deaktiviert werden. „Dauerhaftigkeit der Benutzerdaten“ erlaubt es Webseiten, Benutzerdaten XML-basiert in großem Umfang zu speichern. Diese Speicherstrategie ist eine Alternative zu Cookies. Da sie allerdings nur vom Internet Explorer unterstützt wird, verwenden die meisten Webanwendungen diese Funktionalität nicht. Auf der Windows-Update-Seite wird sie allerdings eingesetzt. Benutzerdaten zu speichern bringt immer eine Unsicherheit mit sich. Deshalb sollte man diese Einstellung auf „Deaktivieren“ setzen, falls man sie nicht benötigt. „Gemischte Inhalte anzeigen“ bestimmt, ob eine sichere Webseite, also eine, die https benutzt, Frames enthalten darf, die Inhalte über http abrufen. Dies macht Seiten unsicher. Leider liefern manche Webanwendungen solche Seiten aus. Deshalb sollte man hier „Bestätigen“ auswählen, wenn man solche Webanwendungen benutzt. Will man sicher gehen, kann man diese Einstellung auch deaktivieren. Dann kann es aber zu fehlerhaften Darstellungen bei Seiten kommen. Der Desktop kann in Windows sogenannte Desktopobjekte beherbergen. Mit „Installation von Desktopobjekten“ bestimmt man, ob Webseiten solche Objekte installieren dürfen. Dies ist sehr gefährlich, da der Desktop und damit das Verhalten der Arbeitsumgebung des Benutzers geändert werden kann. Deshalb sollte man diese Einstellung auf jeden Fall deaktivieren. Will man ein Desktopobjekt von einer Seite installieren lassen, kann man auch „Bestätigen“ wählen. Man sollte jedoch auf keinen Fall hier „Aktivieren“ wählen. Durch „Keine Aufforderung zur Clientzertifikatsauswahl, wenn kein oder nur ein Zertifikat vorhanden ist“ kann man einstellen, dass man keine Eingabeaufforderung erhält, wenn ein Server ein Clientzertifikat erwartet und auf dem Rechner nur eines oder gar keines installiert ist. Es ist sinnvoll diese Einstellung zu deaktivieren, um eine Meldung zu erhalten, falls ein Server ein Zertifikat verlangt. Aber auch eine Aktivierung bringt keine großen Sicherheitsrisiken mit sich. Die Einstellung „Lokaler Verzeichnispfad beim Hochladen von Dateien auf einen Server mit einbeziehen“ sollte man deaktivieren, da der Verzeichnispfad hochgeladener Dateien für einen Webserver irrelevant ist. Ein Verzeichnispfad lässt zusätzliche Informationen durchsickern. So kann z. B. der Windows-Benutzername durch ihn ermittelt werden, falls man eine Datei aus den „Eigenen Dateien“ hoch lädt. Mit „META REFRESH zulassen“ lässt sich bestimmen, ob automatische Weiterleitungen per META-Tag zugelassen werden sollen. Deaktiviert man diese Option, kann die Sicherheit erhöht werden. Ist beispielsweise eine Webanwendung manipuliert worden und soll auf eine zweifelhafte Seite per META-Tag umleiten, wird dies verhindert. Normalerweise zeigen Webseiten einen Link in der Form „klicken Sie hier, falls Sie nicht automatisch weitergeleitet werden“ für Anwender, die eine solche Weiterleitung deaktiviert haben. Ist solch ein Klick für den Anwender zu lästig, kann er auch die Weiterleitung aktivieren. Zwar sinkt damit auch die Sicherheit, aber vor allem, wenn man Skripts aktiviert hat, sind auch andere automatische Weiterleitungen möglich. Wenn man über „Öffnen von Fenstern ohne Adress- und Statusleisten für Webseiten zulassen“ zulässt, dass Fenster ohne Adress- und Statusleisten geöffnet werden können, kann man nicht nachvollziehen, von wo der angezeigte Inhalt stammt und wohin ggf. weitere Links führen. Deshalb sollte man diese Einstellung auf „Deaktivieren“ setzen. Durch „Phishingfilter verwenden“ wird der Einsatz eines Filters zum Erkennen von Phishing-Seiten gesteuert. Dieser sollte aktiviert werden, da er zusätzliche Sicherheit mit sich bringt indem er auf Seiten überprüft, die versuchen Passwörter von Anwendern zu klauen. Der Nachteil liegt in der längeren Ladezeit aufgrund der Überprüfung der Webseite. Über „Popupblocker verwenden“ kann man die Benutzung des Popupblockers regeln. Ein Popupblocker verhindert, dass eine Webseite eigenständig neue Fenster öffnet. Diese Einstellung sollte aktiviert werden, alleine schon wegen der zahlreichen Werbefenster. Der Internet Explorer kann Verzeichnisse darstellen und Dateien ausführen, als würden sie sich lokal auf dem Rechner befinden. Hierzu werden in Firmen

Netzwerkfreigaben genutzt. Die Darstellung erfolgt in einem IFrame und entspricht der des Windows-Explorers. Der IFrame muss dabei eine Referenz auf ein Verzeichnis oder eine Datei in Form einer Netzwerkfreigabe im UNC-Format enthalten. Wie der Internet Explorer sich dabei verhalten soll, kann mit „Programme und Dateien in einem IFRAME starten“ bestimmt werden. Von dieser Einstellung sind auch die Programme und Dateien im Ordner für „Temporäre Internetdateien“ betroffen. Zu diesen gelangt man, wenn man den Button „Einstellungen“ im Bereich Browserverlauf des Reiters „Allgemein“ klickt. Daraufhin öffnet sich ein neues Fenster bei dem ein Klick auf „Dateien anzeigen“ die Temporären Internetdateien anzeigt. Benötigt man den beschriebenen Zugriff nicht, kann man die Einstellung deaktivieren. Möchte man dann trotzdem eine entsprechende Datei öffnen, erhält man eine Meldung. Stellt man „Bestätigen“ ein, wird der Benutzer gefragt, falls eine entsprechende Datei angezeigt oder Anwendung ausgeführt werden soll. „Deaktivieren“ bietet die größte Sicherheit. Ist man sich unsicher, sollte man die Variante mit Eingabeaufforderung wählen. Diese Einstellung auf „Aktivieren“ zu stellen ist nicht ratsam.

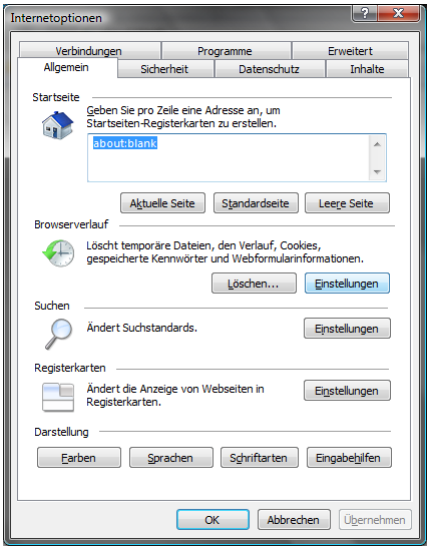

Abbildung 4-12: IE: Internetoptionen – Allgemein

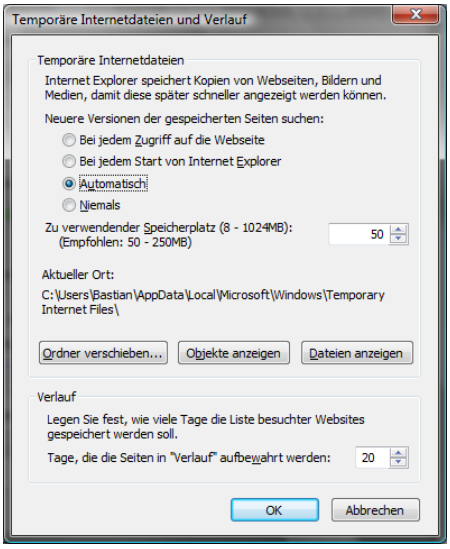

Abbildung 4-13: IE: Temporäre Internetdateien und Verlauf

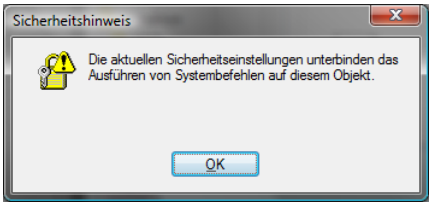

Abbildung 4-14: IE: Meldung – Systembefehl verweigert

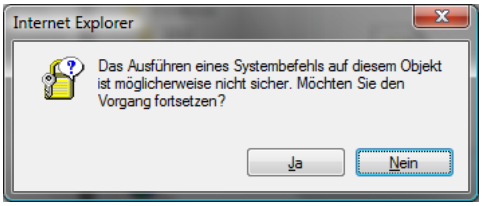

Abbildung 4-15: IE: Meldung – Systembefehl ausführen

Durch „Skript initiierte Fenster ohne Größen- bzw. Positionseinschränkungen zulassen“ kann bestimmt werden, ob ein Skript Fenster beliebig skalieren und an beliebigen Positionen erscheinen lassen kann. Dies ist nicht empfehlenswert, da so auch Browser-Fenster geöffnet werden können, die dem Benutzer nicht auffallen. Deshalb sollte man diese Einstellung deaktivieren. Über „Skripting des Internet-Explorer – Browsersteuerelements zulassen“ kann man Webseiten erlauben, den Internet Explorer fernzusteuern. Dies stellt eine sehr große Gefahr dar, weshalb auch diese Einstellung auf jeden Fall deaktiviert werden sollte. Eine weitere wichtige Einstellung ist „Subframes zwischen verschiedenen Domänen bewegen“. Diese legt fest, ob ein Fenster den Frame eines anderen Fensters verändern kann, auch wenn diese von unterschiedlichen Domänen stammen. Wenn diese

Einstellung aktiviert ist, kann ein Angreifer leicht Webanwendungen, die Frames benutzen, manipulieren. Hierzu muss er nur die ID bzw. den Namen des Frames ermitteln, was keine große Herausforderung darstellt. Danach muss er sein Opfer, das die Webanwendung bereits aufgerufen haben muss, dazu bringen eine vom Hacker kontrollierte Seite anzusurfen. Diese kann nun per Skriptzugriff bestimmen, welcher Inhalt im Frame der Webanwendung, die angegriffen werden soll, dargestellt wird. Wegen der großen Gefährlichkeit sollte man diese Einstellung deaktivieren. Stellt man sie auf „Bestätigen“ erhält man eine Meldung und kann entscheiden ob der domänenübergreifende Zugriff stattfinden soll. Allerdings ist solch ein Zugriff sehr verdächtig. Deshalb ist „Deaktivieren“ hier die richtige Wahl. Andere Browser unterstützen einen solchen Zugriff nicht.

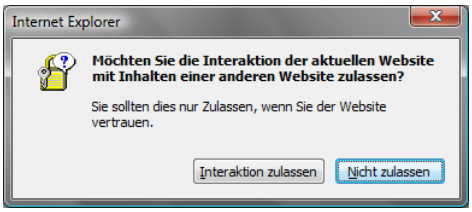

Abbildung 4-16: IE: Meldung – Frame domänenübergreifend ändern

Durch „Unverschlüsselte Formulardaten übermitteln“ wird bestimmt, ob HTML-Formulare über unverschlüsselte Verbindungen gesendet werden dürfen. Da vertrauliche Daten oft über solche Formulare gesendet werden, wäre es sehr gut, zu kontrollieren, ob diese auch über eine gesicherte Verbindung übermittelt werden. Leider werden Formulare aber auch für viele andere Daten, wie Suchanfragen, Gästebucheinträge usw. verwendet. Da diese üblicherweise über ungesicherte Verbindungen gehen, könnte man viele Webseiten nicht mehr benutzen, falls man diese Einstellung deaktiviert.

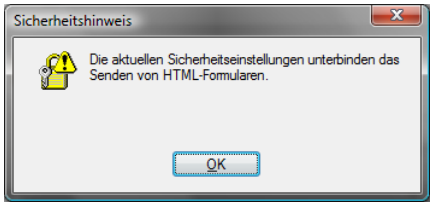

Abbildung 4-17: IE: Meldung – Absenden des Formulars geblockt

Setzt man sie auf „Bestätigen“ wird man sehr viele Meldungen bekommen. Da dies für viele Anwender zu nervig ist, bleibt hier nur die Option „Aktivieren“ übrig. Obwohl dies aus Sicht der Sicherheit nicht gutzuheißen ist. Wäre eine Unterscheidung zwischen POST- und GET-Formularen möglich, so wäre diese Einstellung für POST-Formulare sinnvoller. Je nach Sicherheitszone kann der Internet Explorer die verwendeten Protokolle einschränken. Mit der Einstellung „Verwendung eingeschränkter Protokolle mit aktiven Inhalten für Webseiten zulassen“ kann man bestimmen, ob die Protokolle, die aktive Inhalte verwenden, eingeschränkt werden sollen oder nicht. Da nicht offenliegt, in welcher Weise die Einschränkung des Internet Explorers erfolgt, sollte man die Einstellung auf „Bestätigen“ setzen. Somit kann man im gegebenen Fall selbst entscheiden wie der Browser verfahren soll. Die Einstellung „Websites, die sich in Webinhaltszonen niedriger Berechtigung befinden, können in diese Zone navigieren“ legt fest, ob eine Seite mit weniger Rechten in die aktuelle Zone navigieren kann, z. B. aus Internetzone in die vertrauenswürdige Zone. Um einen Zonenwechsel mitzubekommen, sollte man diese Einstellung auf „Bestätigen“ stellen.

Abbildung 4-18: IE: Meldung – Zonenwechsel

Durch „Ziehen und Ablegen oder Kopieren und Einfügen von Dateien“ kann bestimmt werden, ob Dateien durch Herausziehen oder Kopieren, wie man dies vom Explorer kennt, auf dem Rechner gespeichert werden können. Da somit auch gefährliche Dateien auf den PC gelangen können, sollte man diese Funktion zumindest auf „Bestätigen“ setzen. Noch größere Sicherheit bietet die Option „Deaktivieren“. Mit der Einstellung „Zugriffsrechte für Softwarechannel“ lässt sich festlegen wie Softwarechannels gehandhabt werden sollen. Softwarechannels sind Webseiten, die Programme auf dem Client-PC installieren. Bei der Stufe „Niedrige Sicherheit“ werden die Programme heruntergeladen und installiert und der Benutzer per Mail benachrichtigt. Ist „Mittlere Sicherheit“ eingestellt, wird der Benutzer per Mail benachrichtigt und Programme heruntergeladen, aber nicht installiert. In der Stufe „Hohe Sicherheit“ erfolgt keine Aktion. Da die Programme auch gefährlich sein können, sollte die Option „Hohe Sicherheit“ eingestellt werden.

4.2.2.2 Autovervollständigung

Zu den Einstellungen der Autovervollständigung gelangt man über den Reiter „Inhalte“ innerhalb der Internetoptionen. Im Bereich „AutoVervollständigen“ muss man auf den Button „Einstellungen“ klicken. Danach erscheint ein neues Fenster. Hier kann man bestimmen, ob Eingaben automatisch ergänzt werden sollen. Bei Webadressen, in der Adressleiste ist dies ungefährlich. Bei Formularen können z. B. Kreditkartennummern und ähnliche Informationen gespeichert sein. Diese können dann durch andere Benutzer eingesehen werden, wenn diese die „Pfeil-nach-unten“-Taste im entsprechenden Formularfeld drücken. Deshalb sollte man sich überlegen, ob man diese nicht besser deaktiviert. Werden Benutzernamen und Kennwörter im Browser gespeichert, kann dies problematisch sein. Eine gefährliche Webseite könnte dies nutzen, um einen Benutzer automatisch einzuloggen. Deshalb sollte man die Einstellung „Benutzernamen und Kennwörter für Formulare“ deaktivieren.

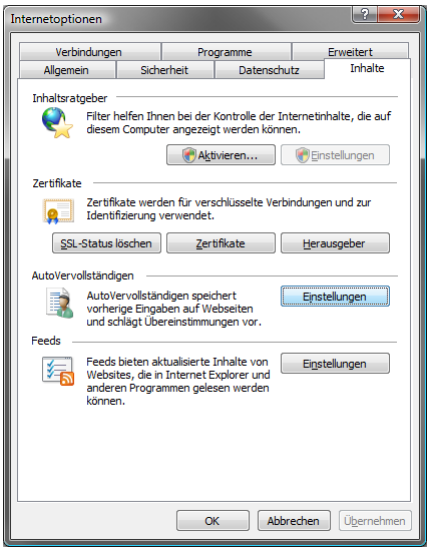

Abbildung 4-19: IE: Internetoptionen – Inhalte

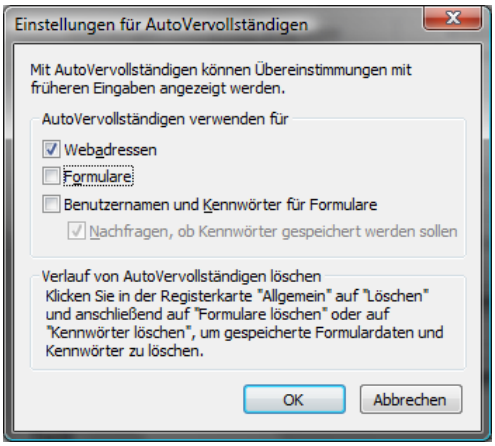

Abbildung 4-20: IE: Einstellungen für Autovervollständigung

4.2.2.3 Add-On-Verwaltung

Die Add-On-Verwaltung kann man über den Reiter „Programme“ innerhalb der Internetoptionen erreichen. Dort klickt man im Bereich „Add-Ons verwalten“ auf die Schaltfläche „Add-Ons verwalten“. Anschließend öffnet sich ein neues Fenster. Hier kann man die installierten Add-Ons deaktivieren bzw. aktivieren oder ggf. löschen. Je weniger Add-Ons aktiviert sind, desto größer ist normalerweise die Sicherheit, wenn nicht gerade das Add-On eine spezielle Sicherheitserweiterung ist. Denn jedes Add-On kann missbraucht werden und somit steigt die Gefahr mit der Anzahl der verwendeten Add-Ons. Welche Add-Ons man benötigt, hängt von den Bedürfnissen des Benutzers ab. Um herauszufinden, welche Add-Ons man benötigt, kann man eines nach dem anderen deaktivieren und danach immer seine Lieblingsseiten ausprobieren. Treten keine unerwünschten Seiteneffekte auf, kann man das Add-On deaktiviert lassen.

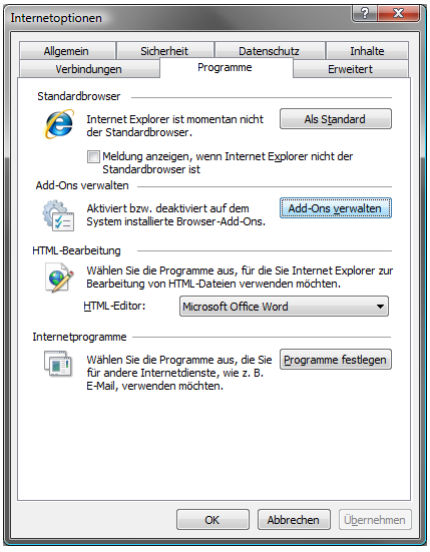

Abbildung 4-21: IE: Internetoptionen – Programme

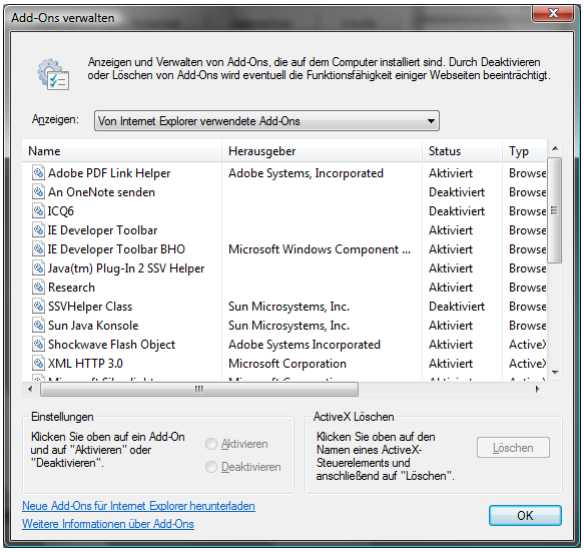

Abbildung 4-22: IE: Add-Ons verwalten

4.2.2.4 Erweiterte Einstellung

Ein paar sicherheitsrelevante Einstellungen finden sich im Reiter „Erweitert“ in den Internetoptionen wieder.

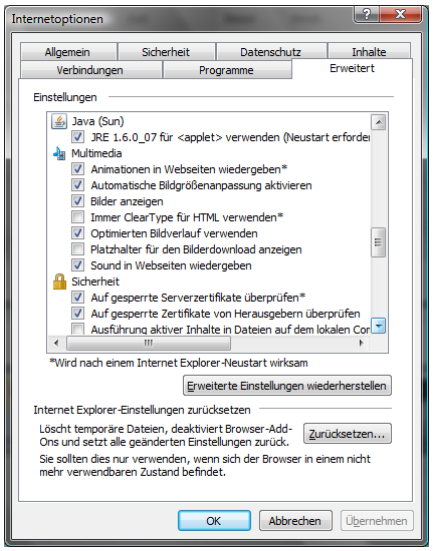

Abbildung 4-23: IE: Internetoptionen – Erweitert

4.2.2.4.1 Java (Sun)

Hier kann man z. B. Java (Sun) deaktivieren bzw. aktivieren. Java ist eine Programmiersprache und wird von manchen Webseiten genutzt, um aktive Inhalte darzustellen. Benötigt man dies nicht, kann man es hier deaktivieren.

4.2.2.4.2 Multimedia

Ist man nur auf Sicherheit bedacht und kann auf jeden Komfort verzichten, können im Bereich „Multimedia“ noch einige Einstellungen vorgenommen werden. Hier kann man das Anzeigen von Bildern über „Bilder anzeigen“ an- oder abschalten. Dies bietet eine zusätzliche Sicherheit, da z. B. CSRF-Attacken IMG-Tags nutzen. Sind Bilder deaktiviert setzt der Browser die entsprechenden GET-Anfragen auch nicht ab. Auch Animationen können ab- bzw. angeschaltet werden. Dies ist über die Einstellung „Animationen in Webseiten wiedergeben“ möglich. So können Fehler in der Rendering-Engine für Animationen nicht ausgenutzt werden.

4.2.2.4.3 Sicherheit

Im Bereich „Sicherheit“ finden sich weitere Einstellungen. Durch „Auf gesperrte Serverzertifikate überprüfen“ und „Auf gesperrte Zertifikate von Herausgebern überprüfen“ werden Zertifikate, die beispielsweise für SSL benötigt werden, zuvor überprüft, um festzustellen ob diese zurückgerufen wurden. Diese Einstellungen sollten aktiviert sein.

Mit „Ausführung aktiver Inhalte in Dateien auf dem lokalen Computer zulassen“ und „Ausführung aktiver Inhalte von CDs auf dem lokalen Computer zulassen“ kann man festlegen, ob aktive Inhalte auf dem lokalen Computer ausgeführt werden sollen. Aktive Inhalte stellen eine Gefahr dar und sollten deshalb nicht aktiviert werden.

Wenn „Beim Wechsel zwischen sicherem und nicht sicherem Modus warnen“ aktiviert ist, erhält der Benutzer jedes Mal eine Benachrichtigung, wenn die Verbindung zwischen sicherem Modus (https) und unsicherem Modus (http) wechselt. Somit bekommt man immer einen solchen Wechsel mit. Weil diese Meldungen dann aber sehr häufig erscheinen, kann man diese Option deaktivieren und sollte dafür bei wichtigen Verbindungen besser auf die Adressleiste achten.

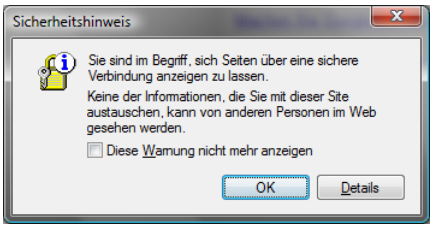

Abbildung 4-24: IE: Meldung – Wechsel von http zu https

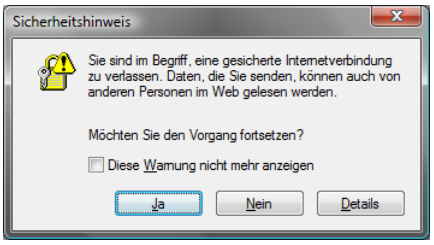

Abbildung 4-25: IE: Meldung – Wechsel von https zu http

Wenn die Einstellung „Installation bzw. Ausführung von Software zulassen, auch wenn die Signatur ungültig ist“ aktiviert wird, wird auch Software mit einer ungültigen Signatur installiert bzw. ausgeführt, was allerdings nicht empfehlenswert ist. Diese Funktion sollte deaktiviert werden, woraufhin der Benutzer eine Eingabeaufforderung erhält, wenn die Signatur von einem Programm ungültig ist.

Über die Einstellung „Integrierte Windows-Authentifizierung aktivieren“ wird ein spezieller Authentifizierungsmechanismus von Microsoft genutzt. Dieser kann nur in der Intranetzone zum Einsatz kommen und verwendet die Benutzerdaten des Windows-Benutzers zur Authentifizierung. Wenn man, wie vorgeschlagen, bei der Benutzerauthentifizierung in den Zoneneinstellungen „Nach Benutzername und Kennwort fragen“ wählt, ist die Einstellung hinfällig. Man kann diese Einstellung noch zusätzlich deaktivieren.

Wird „Leeren des Ordners für temporäre Internetdateien beim Schließen des Browsers“ aktiviert, so werden jedes mal, wenn alle Browserfenster geschlossen werden, die temporären Internetdateien gelöscht, die der Internet Explorer gesammelt hat. Dies kann in so fern zusätzliche Sicherheit bieten, weil diese Dateien dann nicht von anderen Benutzern ausspioniert werden können. Allerdings wird der Internet Explorer dadurch auch langsamer. Je nach Einsatzgebiet sollte man diese Option aktivieren bzw. deaktivieren. Wenn man „Original-XMLHTTP-Unterstützung aktivieren“ einstellt, wird die JavaScript-Implementierung für XMLHTTP-Unterstützung, die z. B. für AJAX nötig ist, verwendet. Deaktiviert man diese Einstellung wird ein ActiveX-Steuerelement von Microsoft dafür verwendet. Da für solche Anfragen auf jeden Fall JavaScript aktiviert werden muss, sollte man sich für die JavaScript-Implementierung entscheiden, da das ActiveX-Steuerelement zusätzliche Risiken birgt. Also ist die Einstellung zu aktivieren. Mit der Einstellung „Phishingfilter“ kann man das Verhalten des Phishingfilters beeinflussen. Die Option „Automatische Websiteprüfung einschalten“ bietet die höchste Sicherheit. Der Filter untersucht hierbei jede Webseite automatisch, wenn er in deren Zone aktiv geschaltet wurde. Bei „Automatische Websiteprüfung ausschalten“ kann der Benutzer manuell Seiten vom Filter überprüfen lassen. Durch „Phishingfilter deaktivieren“ wird die Filterfunktion komplett abgeschaltet. Ist der Filter aktiv, so wird der Internet Explorer zwar langsamer, kann aber den Schutz vor Phishingseiten wesentlich erhöhen. Deswegen ist die Option „Automatische Websiteprüfung einschalten“ empfehlenswert. Die Aktion „Signaturen von heruntergeladenen Programmen überprüfen“ stellt die Integrität eines Programmes und die Identität des Herstellers sicher. Deshalb sollte diese Einstellung eingeschaltet sein. Über „Speicherschutz aktivieren, um das Risiko von Onlineangriffen zu verringern“ kann eine spezielle Schutzfunktion aktiviert werden, die verhindert, dass Code im Speicherbereich des Internet Explorers ausgeführt wird. Dies bringt zusätzliche Sicherheit, die vor den Folgen so genannter Buffer-Overflow-Attacken schützen soll. Allerdings ist diese Einstellung inkompatibel zu manchen Add-Ons und kann zudem nur mit Administratorrechten aktiviert werden. Wenn man nur Add-Ons verwendet, die sich mit dieser

Einstellung vertragen, sollte man sie aktivieren. Über die nächsten drei Einstellungen „SSL 2.0 verwenden“, „SSL 3.0 verwenden“ und „TLS 1.0 verwenden“ kann bestimmt werden, welche Verfahren für SSL-Verbindungen eingesetzt werden dürfen. SSL 2.0 ist bereits veraltet und sollte nicht mehr eingesetzt werden. Die anderen beiden Verfahren gelten als sicher und sollten deshalb aktiviert sein. Über „Verschlüsselte Seiten nicht auf dem Datenträger speichern“ kann man regeln, ob Seiten, die über eine sichere Verbindung empfangen wurden, auf einem Datenträger gespeichert werden dürfen oder nicht. Diese Funktion sollte aktiviert sein. So können solche Seiten nicht lokal gespeichert werden. Andere Benutzer oder Programme könnten sonst Informationen aus diesen Seiten herauslesen. Mit „Warnung anzeigen, wenn die Eingabe in eine Zone umgeleitet wird, in der keine Eingaben zugelassen sind“ kann man das Anzeigen einer Meldung aktivieren, falls Formulardaten auf eine andere Seite umgelenkt werden sollen, die sich nicht in der gleichen Sicherheitszone befindet. Da solche Umleitungen gefährlich sein können, sollte diese Einstellung unbedingt aktiviert sein. Leider wird die Meldung nur bei einem Zonenwechsel und nicht, was besser wäre, bei einem Domänenwechsel angezeigt.

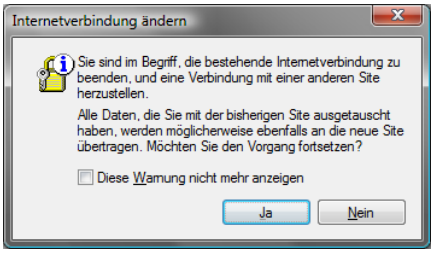

Abbildung 4-26: IE: Meldung – Zonenwechsel von Formulardaten bei Umleitung

Soll der Browser eine „Warnung anzeigen, wenn die Zertifikatadresse nicht übereinstimmt“, muss die gleichnamige Einstellung aktiviert sein. Die Überprüfung des Antragsstellernamens eines Zertifikates wird dadurch eingeschaltet und es erfolgt eine Warnung, wenn der Name der Webseite nicht mit diesem übereinstimmt. Weil gefälschte Zertifikate sonst nicht erkannt werden, ist es wichtig diese Funktion zu aktivieren.

Abbildung 4-27: IE: Meldung – Zertifikatsfehler

4.3 Firefox

Der Firefox benutzt kein Zonenmodell, sondern verwendet für alle Seiten die gleichen Einstellungen. Bei manchen, wie beispielsweise bei „Warnen, wenn Websites versuchen, Add-ons zu installieren“, lassen sich allerdings Ausnahmen definieren. Ich habe zu den standardmäßigen Ausnahmen keine hinzugefügt. Deswegen sind die Testseiten von solchen auch nicht betroffen.

4.3.1 FF mit Standardeinstellungen

Auch der Firefox wird vorab auf die Standardeinstellungen zurückgesetzt. Außerdem sind keine Browsererweiterungen, wie NoScript, installiert. Plugins hingegen, wie Java und Flash, sind bei mir auch im Firefox vorhanden und aktiviert. Die Qualität der Einstellungen des Firefox wird nun wie beim Internet Explorer mit den aufgeführten Methoden überprüft. Die Tests ergaben folgende Ergebnisse.

Negative Ergebnisse:

- Der Browser akzeptiert Applets ohne Nachfrage, auch wenn diese von anderen Domänen stammen. Er macht dabei keine Unterschiede zwischen einzelnen Klassen und JAR-Archiv.
- Der MIME-Typ von Archiven für Applets wird nicht überprüft.
- Cookies sind für Applets per JavaScript les- und schreibbar. Applets können bestehende Cookies für eigene Verbindungen nutzen.
- JavaScript wird kaum eingeschränkt. Folgende Beschränkungen liegen vor:
    - Skripts dürfen nur Fenster schließen, die sie selbst geöffnet haben.
    - Die focus-Methode für Fenster ist nicht erlaubt.
    - Die Adress- und Statusleiste können nicht ausgeblendet werden.
    - Die Fenstergröße kann nicht fixiert werden.
    - Die Statusleiste kann nicht geändert werden.
- Adressen werden nicht auf gefährlichen Code hin überprüft.
- In Frames können Seiten einer anderen Domäne ohne Warnung eingebunden werden, egal ob es sich um einen IFrame oder ein Frameset handelt.
- Der Benutzer wird nicht gewarnt, wenn er unsichtbare Elemente anklickt.
- Links, die auf andere Domänen verweisen, beispielsweise in IMG-Tags, werden nicht gesondert behandelt und ggf. automatisch geladen. Auch die bereits vorhandenen Cookies stehen ihnen zur Verfügung.

Positive Ergebnisse:

- Frame-Bursting funktioniert sowohl bei Framesets als auch bei IFrames.

4.3.2 Einstellungsmöglichkeiten beim FF

Leider ist der Mozilla Firefox ebenso offen für Angriffe auf Webanwendungen bzw. Benutzer, wie der Internet Explorer. Lediglich in den Einschränkungen von JavaScript unterscheiden sich die beiden Browser mit Standardeinstellungen. Das einzig positive Ergebnis ist auch hier, dass Frame-Bursting funktioniert und so ggf. Angriffe wie Clickjacking fehlschlagen. Da also noch einiges an Nachbesserung bei den Sicherheitseinstellungen nötig ist, werde ich nun die Einstellungsmöglichkeiten im Firefox unter Zuhilfenahme von (MozillaZine, 2008; MozillaZine, 2006) erläutern. Es gilt hierbei derselbe Grundsatz wie beim IE: Es gibt nicht die perfekten Einstellungen. Je nach Surfverhalten ist eine Einstellung vielleicht nervend, kann aber mehr Sicherheit bringen oder ist unbedingt nötig. Andere können die Sicherheit in einem Anwendungsfall herabsetzen, dabei aber in anderen Szenarien erhöhen.

Die Einstellungen sind im Firefox über Extras -> Einstellungen… erreichbar. Die sicherheitsrelevanten Einstellungen befinden sich in den Reitern „Sicherheit“, „Erweitert“, „Inhalt“ und „Datenschutz“. Außerdem ist die Add-On-Verwaltung über den Reiter „Allgemein“ erreichbar. Auch diese spielt eine wichtige Rolle bei der Sicherheit des Browsers.

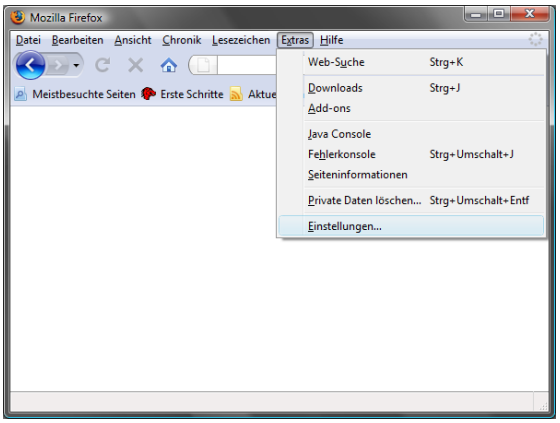

Abbildung 4-28: FF: Extras – Einstellungen…

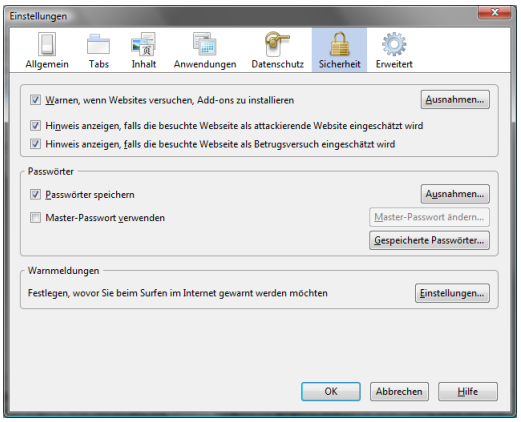

Abbildung 4-29: FF: Einstellungen – Sicherheit

Will man beim Firefox weitere Informationen zu einer Einstellung erhalten, muss man nur auf die Schaltfläche „Hilfe“ klicken. Danach öffnet sich im Browser eine Webseite mit Erklärungen zu den Einstellungen, die im ausgewählten Reiter sichtbar sind. Auch wenn man dort manchmal ein wenig suchen muss, so findet man doch sehr brauchbare Erklärungen.

4.3.2.1 Sicherheit

Im Reiter „Sicherheit“ befinden sich die Einstellungen für Warnmeldungen, Hinweise und Einstellungen zur Passwortverwaltung.

4.3.2.1.1 Wichtige Warnungen und Hinweise

Über „Warnen, wenn Websites versuchen, Add-ons zu installieren“ kann man bestimmen, ob man eine Warnung erhalten soll, wenn eine Seite ein Add-On installieren möchte. Diese Einstellung sollte aktiviert bleiben. Neben der Einstellung befindet sich ein Button „Ausnahmen…“. Über diesen lassen sich in einem neuen Fenster Ausnahmen für Seiten definieren, denen die Installation ohne Warnmeldung erlaubt sein soll, auch wenn die Einstellung aktiviert ist. Der Button ist nur klickbar, wenn „Warnen, wenn Websites versuchen, Add-ons zu installieren“ aktiviert ist.

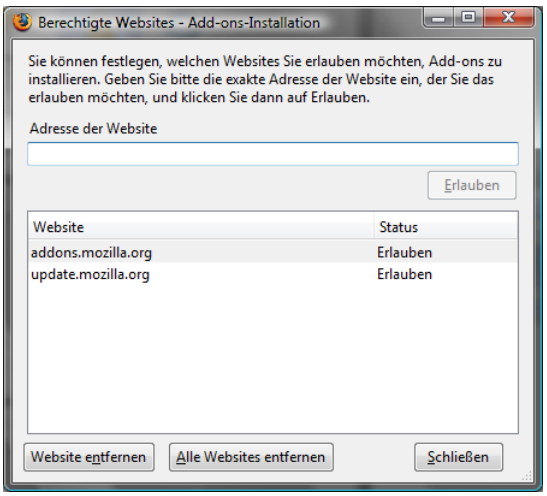

Abbildung 4-30: FF: Add-Ons-Installation – Ausnahmen

Ist die Funktion aktiviert, erhält man eine Meldung, wenn eine Seite versucht, ein nicht verifiziertes Add-On zu installieren und dies geblockt wurde.

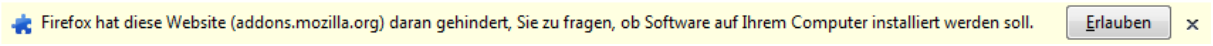

Abbildung 4-31: FF: Meldung – Add-Ons-Installation geblockt

Unabhängig davon, ob für die Seite eine Ausnahme definiert ist, die Einstellung deaktiviert wurde oder der Benutzer die Installation zugelassen hat, kommt vor dieser immer nochmals eine Abfrage.

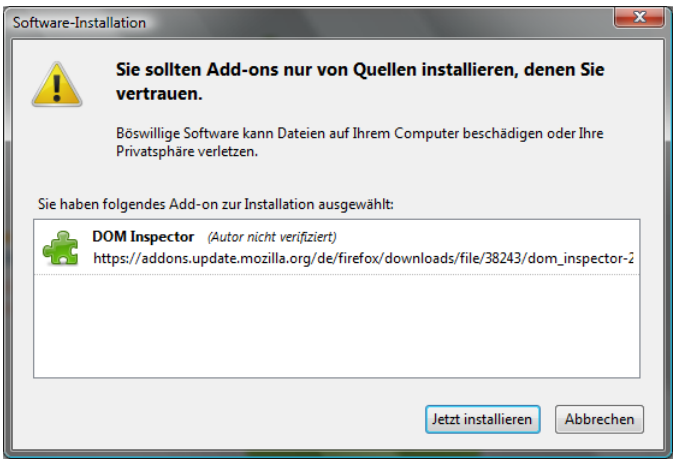

Abbildung 4-32: FF: Meldung – Add-Ons-Installation

Die nächsten beiden Einstellungen beziehen sich auf die integrierte Erweiterung „Google Safe Browsing“ (Google Inc., 2007). Durch „Hinweis anzeigen, falls die besuchte Webseite als attackierende Website eingeschätzt wird“ legt der Benutzer fest, ob Firefox Seiten nach bekannter Malware durchsuchen und bei einem Fund eine Meldung ausgeben soll. Um festzulegen, ob der Browser die aufzurufenden URLs gegenüber einer Blacklist abgleichen und, falls die URL auf der Liste vorhanden ist, eine Meldung anzeigen soll, dient „Hinweis anzeigen, falls die besuchte Webseite als Betrugsversuch eingeschätzt wird“. Diese beiden Einstellungen sollten aktiviert sein.

4.3.2.1.2 Passwörter

Des Weiteren kann man über „Passwörter speichern“ festlegen, ob Firefox Benutzernamen und Kennwörter speichern soll. Wenn man diese Einstellung aktiviert, fragt Firefox bei Eingabe von Benutzernamen mit Passwörtern nach, ob diese gespeichert werden sollen.

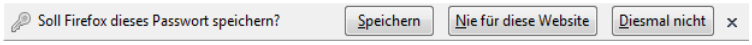

Abbildung 4-33: FF: Meldung – Passwort speichern?

Klickt man auf speichern, so wird das Passwort im Passwortmanager hinterlegt, wählt man „Nie für diese Website“, wird die Seite in die Ausnahmen übernommen. Betätigt man „Diesmal nicht“, wird das Passwort nicht gesichert aber beim nächsten Anmelden wird erneut gefragt, ob man es nun speichern will. Die Liste der Ausnahmen kann man über „Ausnahmen…“ aufrufen und dort Webseiten wieder von der Liste löschen. Der Button ist nur klickbar, wenn „Passwörter speichern“ aktiviert ist.

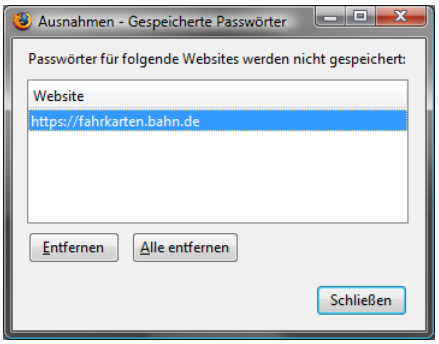

Abbildung 4-34: FF: Meldung – Ausnahme für Passwortspeicherung

Wenn man Passwörter im Browser speichert, bringt dies immer ein Sicherheitsrisiko mit sich. Deshalb sollte darauf verzichtet werden. Wenn die Speicherung aber dennoch erwünscht ist, sollte man auch die nächste Einstellung aktivieren.

Über „Master-Passwort verwenden“ kann man angeben, dass alle Passwörter über ein zusätzliches, frei vergebbares Passwort gesichert werden. Wenn diese Einstellung aktiviert wird, muss man zunächst ein Master-Passwort vergeben. Nur wenn man dieses eingibt, wird der Passwortmanager freigeschaltet. Dies muss man einmal pro Sitzung tun. Sobald der Browser geschlossen und wieder neu geöffnet wird, ist der Passwortmanager zunächst gesperrt.

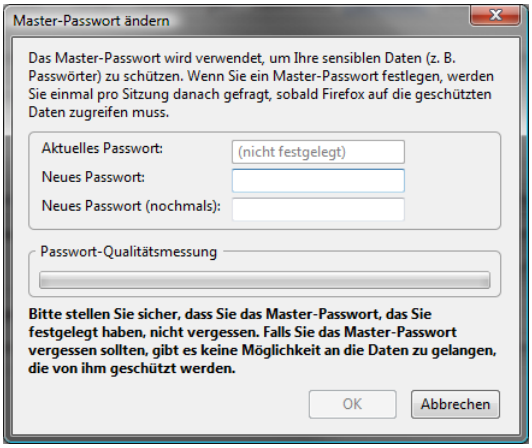

Abbildung 4-35: FF: Meldung – Master-Passwort vergeben / ändern

Die Passwörter sind mit dem Master-Passwort zusätzlich geschützt und können nur nach Eingabe desselben angezeigt oder verwendet werden. Bei aktiviertem Master-Passwort kann man dies über die Schaltfläche „Master-Passwort ändern…“ durch ein Neues ersetzen. Hierbei wird das gleiche Fenster angezeigt, wie bei der Passwortvergabe. Allerdings muss man nun neben dem neuen auch das alte Master-Passwort angeben. Je nachdem, ob das alte Master-Passwort richtig eingegeben wurde oder nicht, erhält der Benutzer eine positive oder negative Rückmeldung.

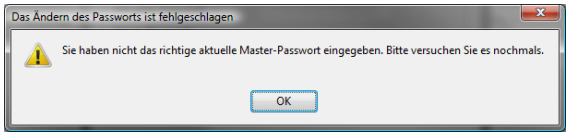

Abbildung 4-36: FF: Meldung – altes Master-Passwort falsch

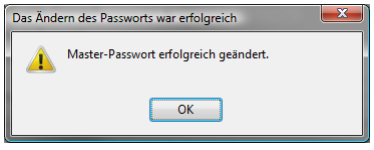

Abbildung 4-37: FF: Meldung – Master-Passwort geändert

Wird eine Aktion durchgeführt, die den Passwort-Manager anspricht, also z. B. das Speichern eines neuen Passworts oder der Aufruf einer Seite, für die ein gespeichertes Passwort benötigt wird, so wird das Master-Passwort verlangt, falls dies für diese Sitzung noch nicht eingegeben wurde.

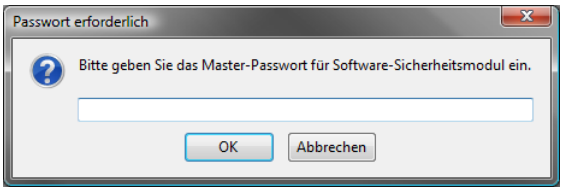

Abbildung 4-38: FF: Meldung – Master-Passwort eingeben

Die Deaktivierung des Master-Passworts ist erst möglich, nachdem man dieses eingegeben hat.

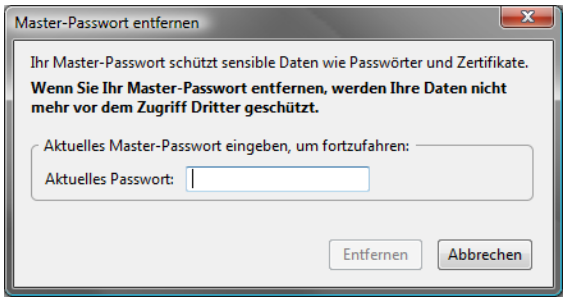

Abbildung 4-39: FF: Meldung – Master-Passwort entfernen

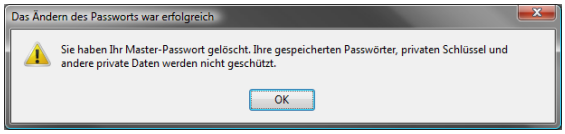

Abbildung 4-40: FF: Meldung – Master-Passwort löschen erfolgreich

Das Master-Passwort wird außerdem auch abgefragt, um über „Gespeicherte Passwörter…“ die Liste der gespeicherten Anmeldedaten anzuzeigen. Hierbei werden die Webseite und der gespeicherte Benutzername angezeigt. Klickt man auf „Passwörter anzeigen“ und ist das Master-Passwort deaktiviert, erfolgt trotzdem eine zusätzliche Eingabeaufforderung, ob man dies wirklich möchte. Bestätigt man dies, so werden auch die Passwörter angezeigt. Ist ein Master-Passwort gesetzt, so muss für diese Aktionen selbiges eingegeben werden.

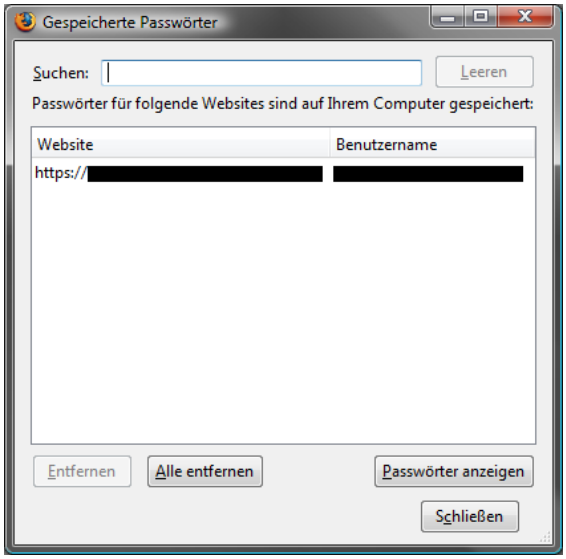

Abbildung 4-41: FF: gespeicherte Benutzerdaten

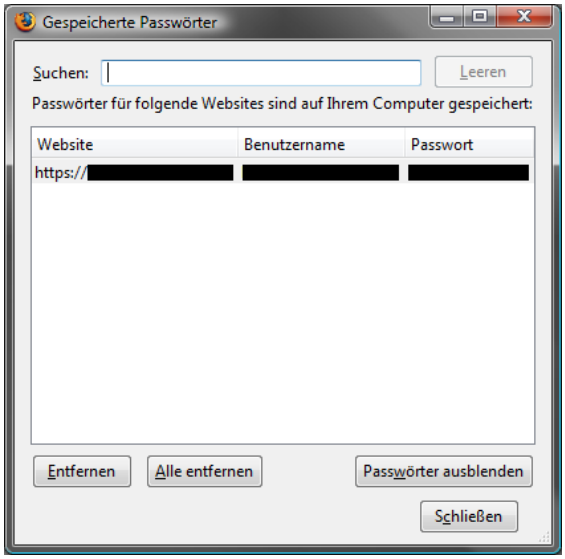

Abbildung 4-42: FF: gespeicherte Benutzerdaten inkl. Passwörter

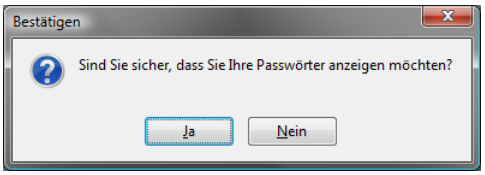

Abbildung 4-43: FF: Meldung – Passwörter anzeigen?

4.3.2.1.3 Weitere Warnmeldungen

Über den Button „Einstellungen…“ im Bereich Warnmeldungen, können Einstellungen zu weiteren Warnmeldungen vorgenommen werden. Es erscheint ein neues Fenster, bei dem man auswählen kann, ob eine Warnung für die jeweilige Einstellung angezeigt werden soll. Allen Sicherheitswarnungen ist dabei „Eine Sicherheitswarnung anzeigen, wenn“ vorangestellt.

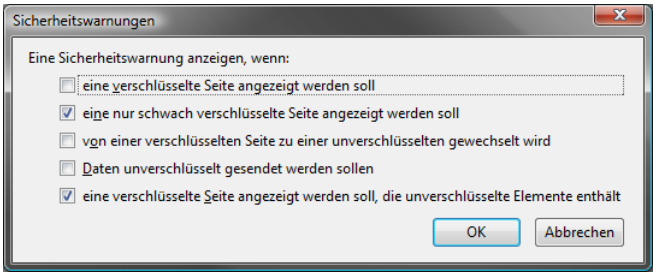

Abbildung 4-44: FF: Sicherheitswarnungen

Bei „eine verschlüsselte Seite angezeigt werden soll“ wird eine Meldung ausgegeben, wenn eine sichere Verbindung (https) zu einer Seite aufgebaut wird. Dies ist weniger sinnvoll, da man schnell von den Meldungen genervt ist. Besser ist es, die Adressleiste zu überwachen, wenn man wichtige Seiten besucht. Setzt man das Häkchen bei „eine nur schwach verschlüsselte Seite angezeigt werden soll“, so warnt Firefox, vor der Benutzung einer schwach verschlüsselten https-Seite. Dies sollte aktiviert bleiben.

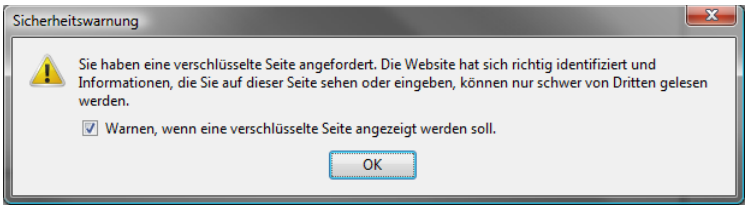

Abbildung 4-45: FF: Meldung – sichere Seite

Durch „von einer verschlüsselten Seite zu einer unverschlüsselten gewechselt wird“ wird beim Wechsel von einer https- zu einer http-Verbindung gewarnt. Mittels „Daten unverschlüsselt gesendet werden sollen“ wird gewarnt, wenn man unverschlüsselte Formulare absendet. Diese beiden Einstellungen sind sinnvoll, führen aber auch zu vielen Meldungen. Deshalb ist es meist besser, sie zu deaktivieren und dafür auf https-Verbindungen zu achten.

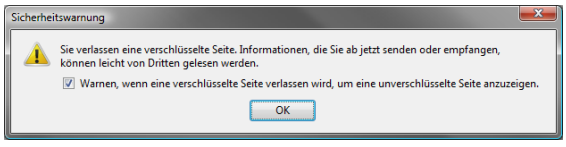

Abbildung 4-46: FF: Meldung – verschlüsselte Seite verlassen

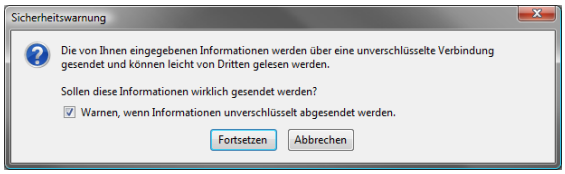

Abbildung 4-47: FF: Meldung – unverschlüsselte Formulardaten übermitteln

Bei „eine verschlüsselte Seite angezeigt werden soll, die unverschlüsselte Elemente enthält“ erhält man eine Warnung, falls eine sichere Seite (https) unsichere Elemente (http), z. B. in einem IFrame, enthält. Dies ist wichtig und sollte aktiviert werden. Firefox warnt normalerweise, wenn eine verschlüsselte Seite unverschlüsselte Daten übermittelt. Enthält eine https-Seite allerdings einen Frame der eine http-Verbindung verwendet, können aus diesem ohne Warnung unverschlüsselte Daten gesendet werden. Deshalb sollte man diese Einstellung aktivieren, um gewarnt zu werden, falls unsichere Elemente in einer sicheren Verbindung eingebunden werden.

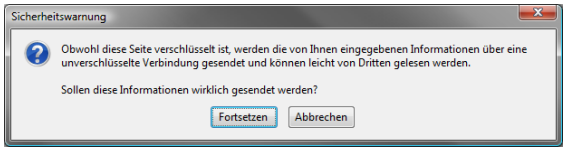

Abbildung 4-48: FF: Meldung – unverschlüsselte Daten aus einer https-Seite senden

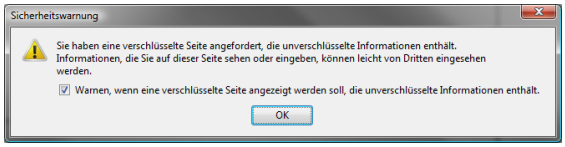

Abbildung 4-49: FF: Meldung – https-Seite enthält http-Elemente

Außerdem warnt Firefox, falls POST-Daten umgelenkt werden sollen.

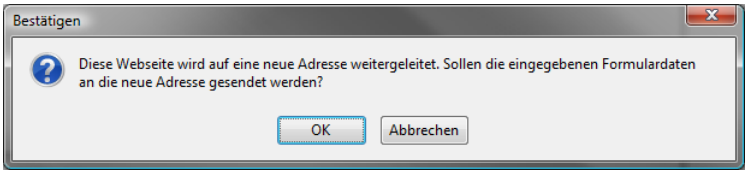

Abbildung 4-50: FF: Meldung – POST-Daten werden umgelenkt

4.3.2.2 Erweitert

Der Reiter „Erweitert“ beherbergt selbst wieder einige Reiter. Von diesen sind für Sicherheitsaspekte die Reiter „Allgemein“ und „Verschlüsselung“ relevant.

4.3.2.2.1 Allgemein

Im Reiter „Allgemein“ kann man mit der Einstellung „Warnen, wenn Websites versuchen umzuleiten oder neuzuladen“ einstellen, ob META-REFRESH-Tags vom Firefox ausgeführt werden sollen oder nicht. Aktiviert man diese Funktion, erhöht es die Sicherheit.

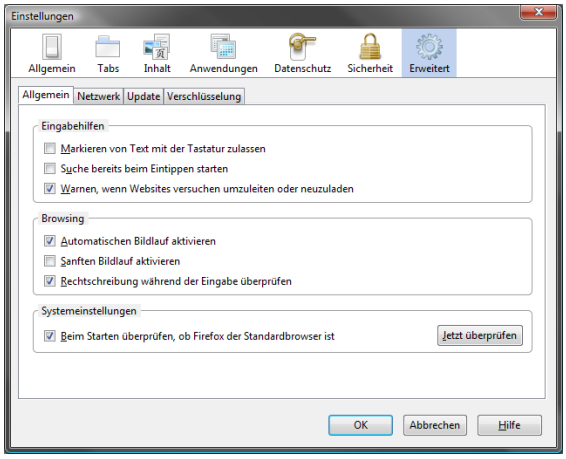

Abbildung 4-51: FF: Einstellungen – Erweitert – Allgemein

Wenn eine Seite einen META-REFRESH-Tag enthält, zeigt Firefox bei aktivierter Einstellung eine Warnmeldung an. So kann man bestimmen, ob der Tag ausgeführt werden soll oder nicht.

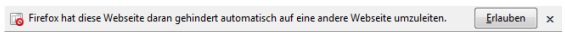

Abbildung 4-52: FF: Meldung – META REFRESH blockiert

4.3.2.2.2 Verschlüsselung

Im Reiter „Verschlüsselung“ kann bestimmt werden, welche Verschlüsselungsprotokolle Firefox verwenden und wie er mit Zertifikaten umgehen soll. Die Protokolle „SSL 3.0“ und „TLS 1.0“ gelten

als sicher und sollten deshalb aktiviert bleiben. Eine weitere sicherheitsrelevante Funktion bietet die Zertifikat-Validierung, die über „Validierung“ erreicht werden kann. Hier kann man einstellen, ob der Zertifikatsstatus über OCSP erfragt werden soll, falls dies möglich ist. Dies sollte eingeschaltet werden, um zurückgerufene Zertifikate zu erkennen. Die Einstellung „Wenn eine OSCP-Server-Verbindung fehlschlägt, das Zertifikat als ungültig betrachten“ weist Zertifikate, die einen OSCP-Server enthalten ab, wenn zu diesem keine Verbindung möglich ist. Dies erhöht die Sicherheit zusätzlich, kann aber zu Problemen führen, wenn die Kommunikation mit einem OSCP-Server dauerhaft fehlschlägt. Der Benutzer sollte die Funktion aktivieren und nur deaktivieren, falls er eine Beeinträchtigung feststellt.

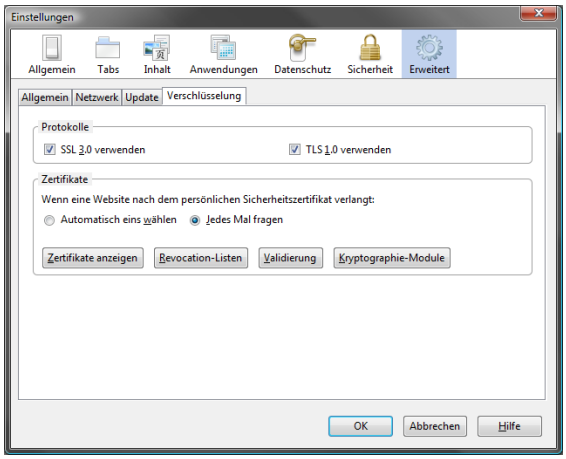

Abbildung 4-53: FF: Einstellungen – Erweitert – Verschlüsselung

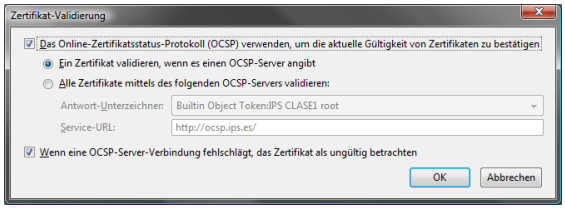

Abbildung 4-54: FF: Zertifikat-Validierung

Firefox warnt zusätzlich vor fehlerhaften Zertifikaten. Außerdem zeigt er eine Fehlermeldung an, falls ein Protokoll deaktiviert wurde, das zur Kommunikation nötig ist.

Abbildung 4-55: FF: Meldung – Zertifikatsfehler

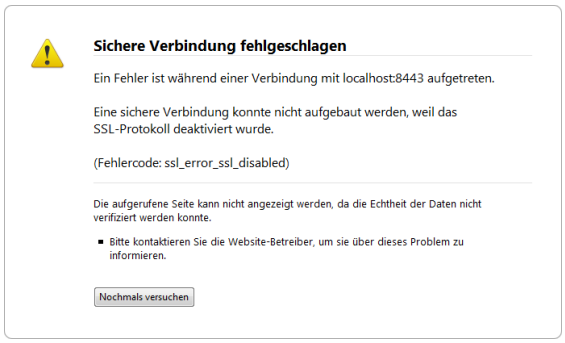

Abbildung 4-56: FF: Meldung – Protokoll deaktiviert

4.3.2.3 Inhalt

Einstellungen für Popups, Grafiken, JavaScript und Java, findet man im Reiter „Inhalt“.

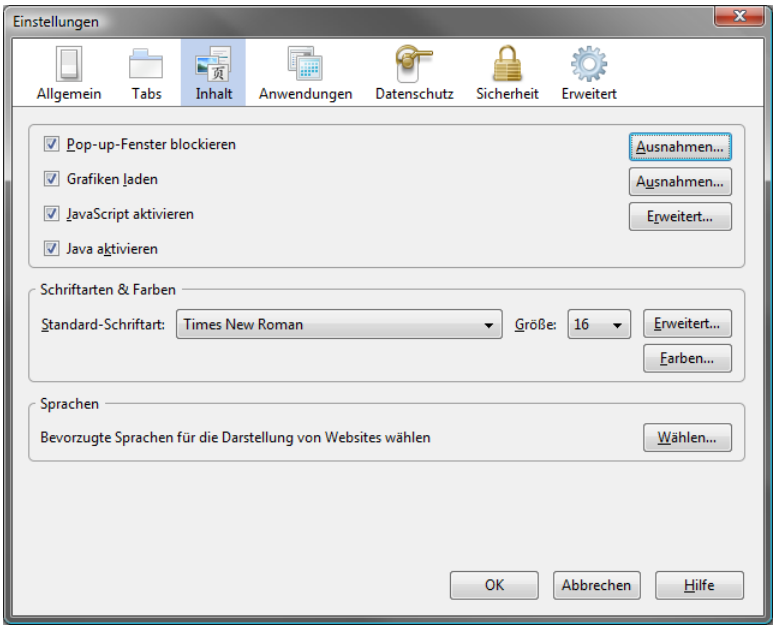

Abbildung 4-57: FF: Einstellungen – Inhalt

4.3.2.3.1 Popup-Blocker

Den Popup-Blocker sollte man aktiviert lassen. Erlaubt man bestimmten Seiten Popups anzuzeigen, findet man dies in der Ausnahmeliste. Man kann hier neue Seiten hinzufügen und bestehende löschen. Die Ausnahmeliste ist über die Schaltfläche „Ausnahmen…“ neben der entsprechenden Einstellung erreichbar. Ist der Popup-Blocker deaktiviert, kann man auch nicht auf die Ausnahmeliste zugreifen.

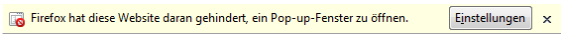

Abbildung 4-58: FF: Meldung – Popup blockier

Man kann mit einem Klick auf „Einstellungen“ Popups für die angezeigte Seite dauerhaft zulassen, das Popup einmalig anzeigen, direkt zur Ausnahmeliste navigieren oder die Anzeige für blockierte Popups deaktivieren.

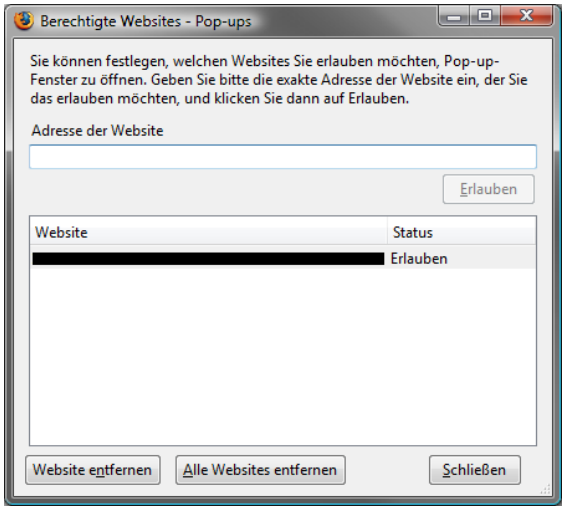

Abbildung 4-59: FF: Ausnahmen für Popup-Blocker

4.3.2.3.2 Grafiken

Mit der Einstellung „Grafiken laden“, kann man bestimmen, ob Bilder geladen werden sollen. Da man normalerweise Bilder ansehen möchte, sollte diese Einstellung aktiviert bleiben. Ist sie deaktiviert, gewinnt man minimal an Sicherheit, da z. B. auch CSRF-Attacken, die das src-Attribut von Bildern verwenden, fehlschlagen würden. Auch hier können Ausnahmen für Seiten definiert werden. Es ist möglich, sowohl bei solchen Grafiken zu erlauben oder zu blockieren. Deshalb sind die Ausnahmen über die nebenstehende Schaltfläche auch immer erreichbar, egal ob Grafiken generell geladen werden oder nicht.

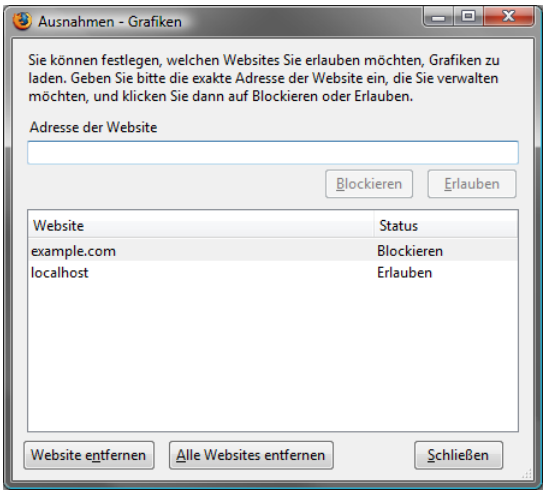

Abbildung 4-60: FF: Ausnahmen für zu ladende Grafiken

4.3.2.3.3 JavaScript

Mit der nächsten Einstellung kann man JavaScript an- oder abschalten. Ist JavaScript aktiviert, können durch einen Klick auf „Erweitert…“ weitere Einstellungen vorgenommen werden. JavaScript ist ein großes Sicherheitsrisiko, da es von vielen Angriffen benutzt wird, die ohne JavaScript unmöglich wären. Allerdings bauen auch viele Webanwendungen auf JavaScript auf, deshalb muss der Benutzer selbst entscheiden, was hier die richtige Einstellung ist. Hat man JavaScript aktiviert, sollte man sich die erweiterten Einstellungen für JavaScript ansehen. „Existierende Fenster verschieben oder deren Größe ändern“ lässt zu, dass JavaScripts die Größe bzw. Position eines Fensters ändern, dies kann gefährlich sein und sollte deaktiviert werden. Ob die focus()-Methode verwendet werden darf, legt „Fenster vor oder hinter andere Fenster legen“ fest. Auch dies sollte verhindert werden. Wenn ein JavaScript das Kontextmenü ersetzt, kann dies Benutzer verwirren. Deshalb sollte auch „Das Kontextmenü deaktivieren oder ersetzen“ abgeschaltet werden. Auch das Ausblenden der Statusleiste „Statusleiste ausblenden“ und deren Änderung „Statusleiste ändern“ sollte nicht erlaubt sein.

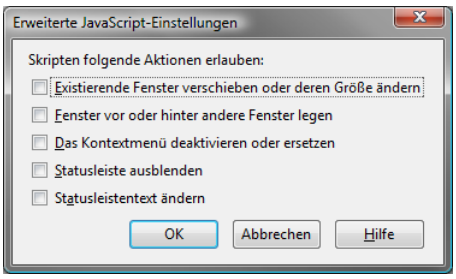

Abbildung 4-61: FF: Erweiterte JavaScript-Einstellungen

4.3.2.3.4 Java

Die letzte sicherheitsrelevante Einstellung in diesem Reiter ist, ob Java an- oder abgeschaltet werden soll. Benötigt man kein Java, kann man es deaktivieren und so GIFAR-Attacken abwehren. Nutzt man aber eine Webanwendung, die Java verwendet, z. B. einen Chat, der auf Java aufbaut, so muss man zwangsläufig Java aktivieren.

4.3.2.4 Datenschutz

Auf die Anwendungssicherheit bezogen, ist die Einstellung „Daten speichern, die in Formulare und die Suchleiste eingegeben werden“ relevant. Da in Formularen auch Daten, wie

Kreditkartennummern und Benutzernamen eingegeben werden, kann dies kritisch sein, falls auch andere Benutzer dasselbe Profil verwenden können. Auch wenn sie „nur kurz einmal etwas im Internet nachschauen“ müssen. Denn dadurch können sie solche Daten ausspähen. Deshalb sollte diese Einstellung aus Sicherheitsgründen deaktiviert werden, wenn man auf den Komfort verzichten kann. Außerdem sollte man „Private Daten löschen, wenn Firefox beendet wird“ auswählen, so werden alle wichtigen Daten, die der Browser gesammelt hat, gelöscht. Setzt man zusätzlich das Häkchen bei „Vor dem Löschen von privaten Daten fragen“, erscheint eine Eingabeaufforderung, welche Daten man löschen möchte. Die privaten Daten lassen sich auch über „Jetzt löschen“ im Reiter Datenschutz entfernen.

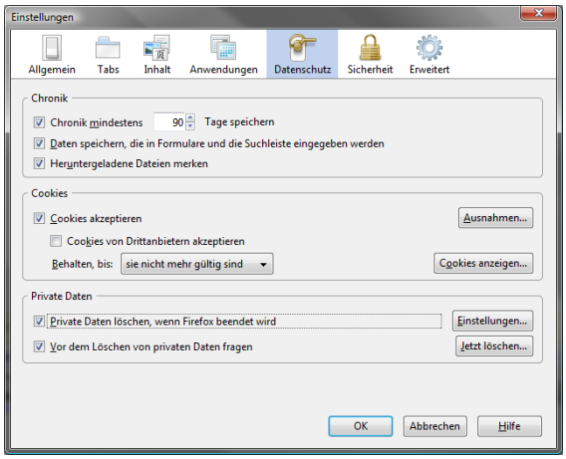

Abbildung 4-62: FF: Einstellungen – Datenschutz

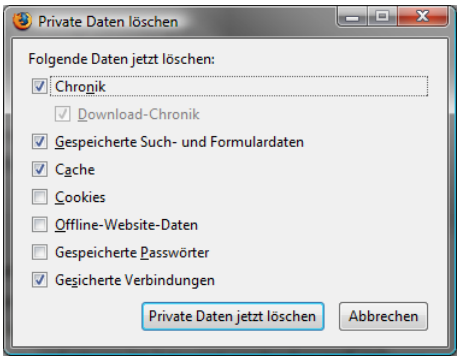

Abbildung 4-63: FF: Private Daten löschen

4.3.2.5 Add-On-Verwaltung

Die Add-On-Verwaltung ist über den Button „Add-ons verwalten“ im Reiter „Allgemein“ erreichbar. Wichtig für die Sicherheit ist hier der Bereich „Plugins“. Dieser kennzeichnet Software, die benutzt wird, um Inhalte darzustellen, die der Browser selbst nicht interpretieren kann. Dies sind z. B. Java, PDF-Dokumente, Flash usw. Nicht benötigte Plugins sollten deaktiviert werden, da jedes Plugin neue Sicherheitsrisiken mit sich bringt.

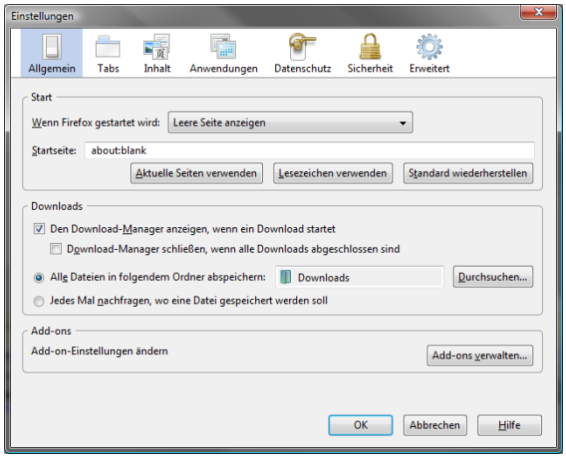

Abbildung 4-64: FF: Einstellungen – Allgemein

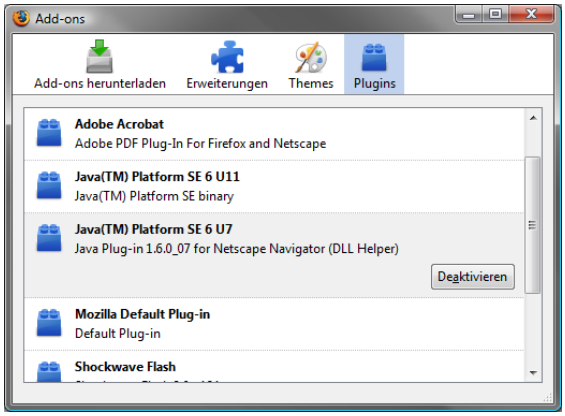

Abbildung 4-65: FF: Add-On-Verwaltung

5 Untersuchte Plugins

Da es mit der Browsersicherheit nicht gut bestellt ist, ist man gut beraten, zusätzliche Sicherheitsplugins zu nutzen. Hiermit sind Plugins im Sinne von Browsererweiterungen gemeint, also solche, die die Funktionalität desselben ändern. Für den Internet Explorer findet man diese unter (Microsoft Corporation, 2008), während sie für den Firefox unter (Mozilla Foundation, 2008) aufgeführt sind. Mit „Controle de Scripts“ und „NoScript“ stelle ich nachfolgend zwei Plugins vor, die die Sicherheit des Firefox erhöhen. Ich wählte keines für den Internet Explorer aus, da es derzeit für diesen nur Sicherheitsplugins gibt, die die Browsersicherheit, jedoch nicht die Daten- und Anwendungssicherheit erhöhen.

5.1 Controle de Scripts

Die erste Browsererweiterung „Controle de Scripts“ in der Version 1.0.3 (Mozilla Foundation, 2008) ermöglicht eine bessere Einschränkung von JavaScript. Neben den standardmäßig vorhandenen Einstellungen erhält man nun weitere, wenn man die erweiterten JavaScript Einstellungen anzeigen lässt, siehe [Abbildung 4-57: FF: Einstellungen – Inhalt – Seite 74] bzw. [Abbildung 4-61: FF: Erweiterte JavaScript-Einstellungen – Seite 75]. Die Berechtigungen von JavaScript werden dabei um Einstellungen zum Verändern von Grafiken, Schließen von Fenstern, Ändern der Größe von Fenstern, sowie dem Ausblenden von Menüleiste, Navigations-Symbolleiste, Adressleiste, Lesezeichen-Symbolleiste und Scroll-Leisten erweitert.

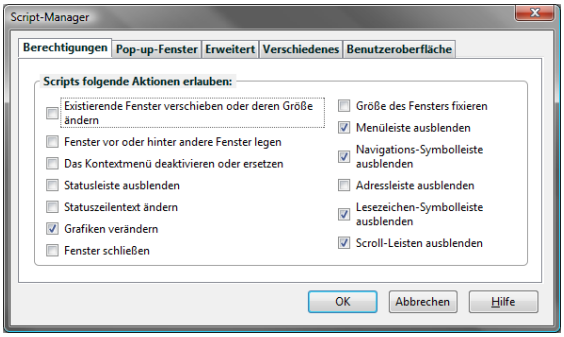

Abbildung 5-1: Controle de Scripts: JavaScript-Berechtigungen

Im Reiter „Pop-up-Fenster“ wird festgelegt, bei welchen Ereignissen ein Popup zulässig ist.

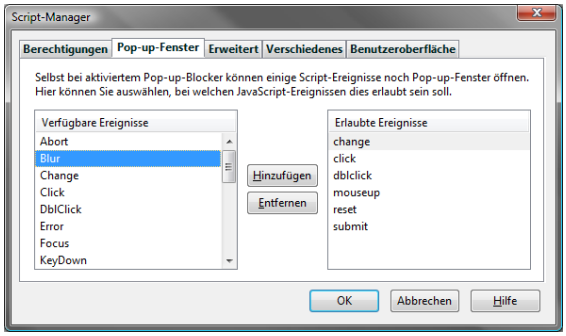

Abbildung 5–2: Controle de Scripts: Popups

Weiß man nicht, wann ein Ereignis eintritt, kann man sich dies per Rechtsklick in einem neuen Fenster anzeigen lassen.

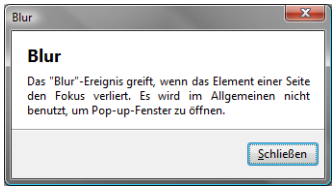

Abbildung 5–3: Controle de Scripts: Ereignisbeschreibung

Fortgeschrittene Benutzer können im Reiter „Erweitert“ den Zugriff auf beliebige Eigenschaften oder Methoden festlegen. Man kann bei „Hinzufügen“ eine der folgenden Regeln auswählen: „Erlauben“, „Blockieren“ oder „Gleicher Ursprung“ (von derselben Seite aus). Legt man den Zugriff auf eine

Eigenschaft fest, kann man zudem zwischen „Nur lesen“, „Nur schreiben“ oder „Lesen & schreiben“ wählen. Wichtig ist, dass man als Objekt-Namen die DOM-Bezeichnung verwendet, also beispielsweise „HTMLDocument“ für das JavaScript-Objekt „document“. Standardmäßig wird die hinzugefügte Regel der Standardsicherheitsrichtlinie „Default“ hinzugefügt, die für alle Seiten gilt. Allerdings kann man auch über die Schaltfläche „Neu“ eigene Sicherheitsrichtlinien erzeugen, für die man separate Regeln festlegen kann. Zudem ist es möglich, einer Richtlinie bestimmte Webseiten zuzuordnen. Hierbei nutzt „Controle de Scripts“ eine eingebaute Funktionalität des Firefox (MozillaZine, 2007; Ruderman, 2006). Die Zuordnung der Webseiten ist durch einen Klick auf „Website-Liste“ möglich. Hierzu öffnet sich ein neues Fenster. Eigens angelegte Sicherheitsrichtlinien besitzen Vorrang vor der Standardrichtlinie.

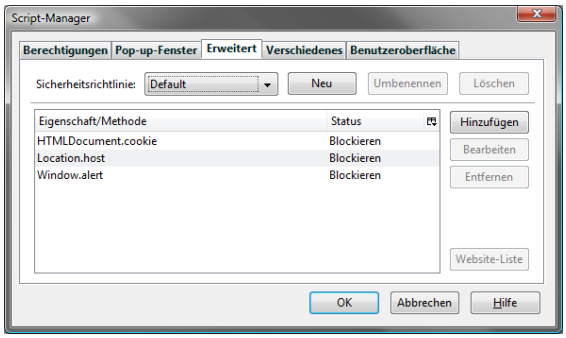

Abbildung 5–4: Controle de Scripts: Erweitert

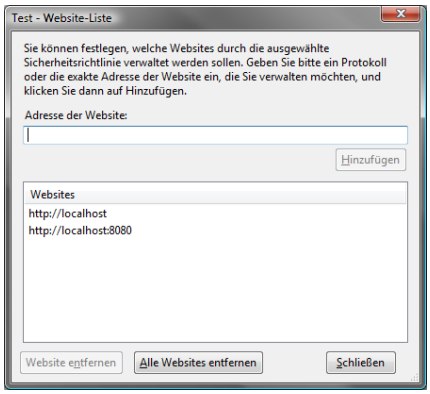

Abbildung 5–5: Controle de Scripts: Website-Liste

Leider hat die Erweiterung noch Probleme beim Löschen von Regeln. Außerdem greifen die selbst erstellten Sicherheitsrichtlinien nicht, wenn man NoScript verwendet.

5.2 NoScript

Ein sehr interessantes Sicherheitsplugin ist „NoScript“ in der Version 1.8.8.8 (Mozilla Foundation, 2009). Durch seinen großen Funktionsumfang und die kurzen Reaktionszeiten des Entwicklerteams ist es sehr empfehlenswert. Bereits kurz nachdem Clickjacking in die öffentliche Diskussion geraten war, wurden diesbezügliche Abwehrmaßnahmen in NoScript implementiert (Maone, 2008). Zu den Einstellungen von „NoScript“ gelangt man über einen Klick auf das Plugin-Symbol in der Statusleiste, das sich je nach Status leicht ändern kann: Verwendet die angezeigte Seite JavaScript, ist das „S“ blau, ansonsten weiß. Wird kein weiteres Symbol angezeigt, ist die Seite nicht eingeschränkt. Ein Ausrufzeichen signalisiert, dass JavaScript global erlaubt ist. Ist der Hintergrund zur Hälfte schwarz, sind nur nicht vertrauenswürdige Skripte blockiert. Bei einem Verbotszeichen über dem „S“ sind alle Skripte verboten. Ist das Verbotszeichen in der rechten unteren Ecke, sind Skripte auf der Seite teilweise erlaubt.

Im erscheinenden Menü hat man Zugriff auf die wichtigsten Funktionen. Klickt man auf Einstellungen, werden diese angezeigt.

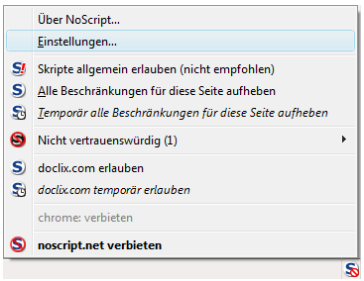

Abbildung 5–6: NoScript: Symbolmenü

Im Reiter „Allgemein“ befinden sich Einstellungen, die sich auf alle Seiten auswirken. Mit der Einstellung „Jeweils aktuelle Top-Level-Site temporär erlauben“ fügt man die aktuell aufgerufene Seite zu den temporär erlaubten Seiten hinzu, was jedoch nicht empfehlenswert ist. Die nächste Einstellung bezieht sich auf die NoScript-Symbolleiste. Man kann mit ihr bestimmen, was bei einem Linksklick geschehen soll. Betroffene Seiten nach einer Berechtigungsänderung zu laden ist sinnvoll, deshalb sollte die entsprechende Einstellung aktiv sein. Des Weiteren kann festlegt werden, ob Lesezeichen als sichere Webseiten gelten und ob man JavaScript allgemein aktivieren will. Dies ist nicht ratsam. Im Reiter „Positivliste“ erkennt man alle Webseiten, die derzeit als vertrauenswürdig gelten. Temporär vertrauenswürdige Seiten sind kursiv.

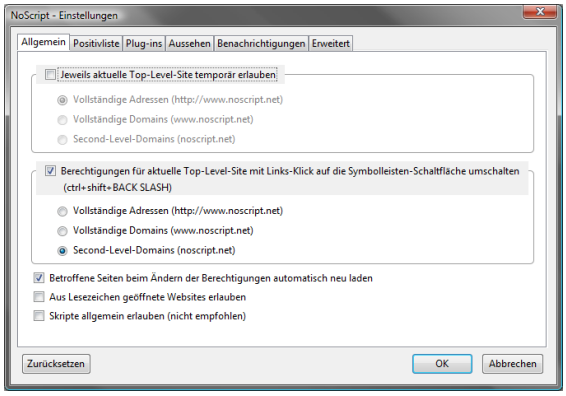

Abbildung 5–7: NoScript: Allgemein

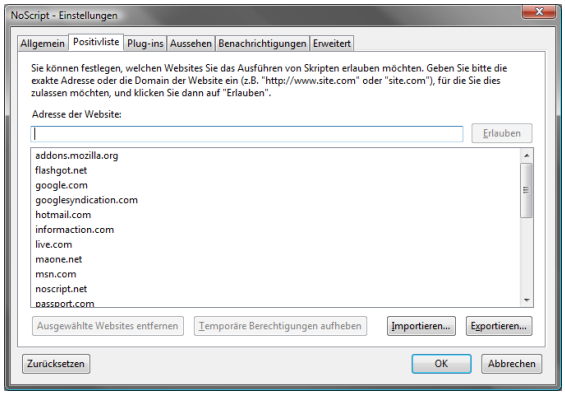

Abbildung 5–8: NoScript: Positivliste

Über den Reiter „Plug-ins“ lassen sich die verschiedenen Plugins aktivieren bzw. deaktivieren. Zudem lässt sich bestimmen, inwiefern für deaktivierte Plugins Platzhalter angezeigt werden sollen und ob vor dem temporären Anzeigen eines Objekts bei einem Klick eine Bestätigungsmeldung zu sehen sein soll. Ich empfehle, alle Plugins zu deaktivieren und beim Klicken auf einen Platzhalter die Objekte erst nach einer Bestätigungsmeldung anzeigen zu lassen. Des Weiteren lassen sich Frames und IFrames verbieten, dies sollte man auch tun. Außerdem ist das Aktivieren einer Schutzfunktion gegen Clickjacking, der „ClearClick-Schutz“ möglich und es kann bestimmt werden, ob unsichtbare Objekte auf Seiten sichtbar gemacht werden sollen. Man sollte den ClearClick-Schutz für nicht vertrauenswürdige und vertrauenswürdige Seiten aktivieren. Das Anzeigen von unsichtbaren Objekten ist eher Geschmackssache. Es empfiehlt sich aber, dies für nicht vertrauenswürdige Seiten anzuschalten.

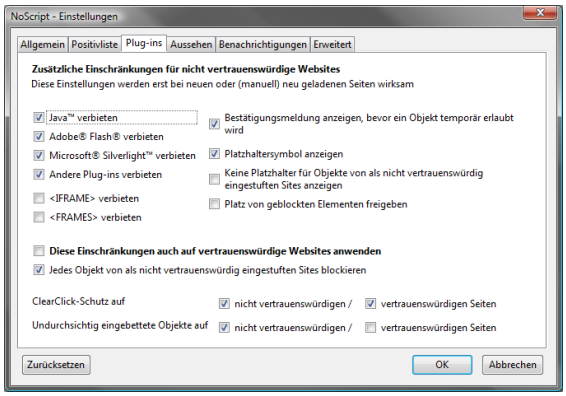

Abbildung 5–9: NoScript: Plug-Ins

Abbildung 5–10: NoScript: Platzhalter

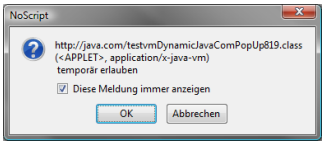

Abbildung 5–11: NoScript: Meldung – blockiertes Objekt anzeigen

Der letzte Reiter mit sicherheitsrelevanten Einstellungen ist „Erweitert“. Hier befinden sich weitere Kategorien. In der Kategorie „Nicht vertrauenswürdig“ lassen sich weitere Einstellungen für nicht vertrauenswürdige Seiten festlegen. Entsprechend gibt es auch eine Kategorie „Vertrauenswürdig“ für vertrauenswürdige Seiten. Nachfolgend sind die wichtigsten Begriffe dieser Einstellungen erläutert:

- Bookmarklets sind Lesezeichen, die mit „javascript:“ beginnen und somit JavaScript ausführen.
- Bei einem Klick auf einen a- oder area-Tag ruft Firefox im Hintergrund zusätzlich die URL des PING-Attributes auf, falls dieses vorhanden ist.
- NOSCRIPT-Tags werden normalerweise vom Browser angezeigt, falls JavaScript deaktiviert ist.
- Web-Bugs sind Bilder, die 1x1 Pixel groß und nicht sichtbar sind oder die gleiche Farbe wie der Hintergrund besitzen und der Überprüfung des Aufrufs einer Seite dienen.
- META-Weiterleitungen werden zumeist in NOSCRIPT-Tags verwendet um den Anwender auf eine JavaScript-freie Variante der Seite zu leiten.
- JavaScript-Links sind URLs die mit „javascript:“ beginnen.

Die Standardeinstellungen bieten hier bereits eine große Sicherheit. Verbietet man alles, kann dies die Sicherheit weiter erhöhen. Vor allem das PING-Attribut kann von Angreifern missbraucht werden.

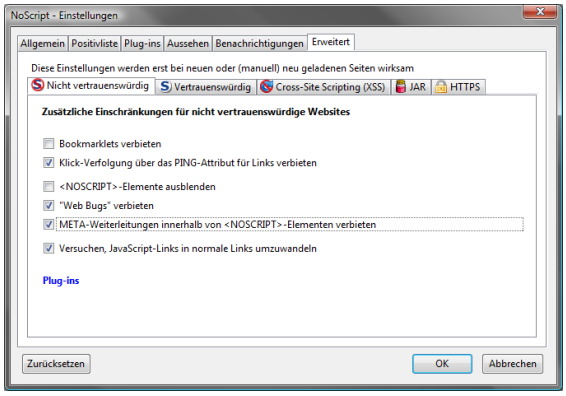

Abbildung 5–12: NoScript: Erweitert – nicht vertrauenswürdige Seiten

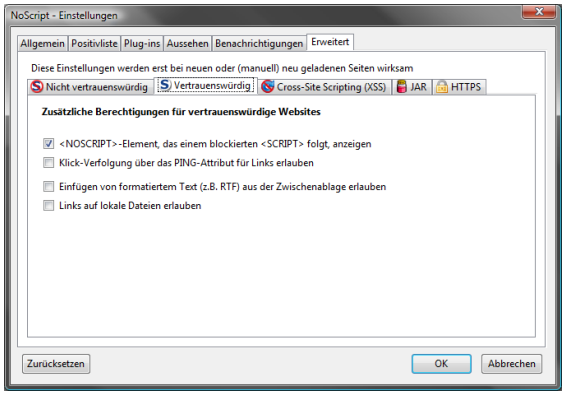

Abbildung 5–13: NoScript: Erweitert – vertrauenswürdige Seiten

„NoScript“ schützt Anwender auch vor XSS-Angriffen. Hierfür werden Anfragen auf Muster hin durchsucht. Erkennt das Plugin eine XSS-Attacke, warnt es den Benutzer.

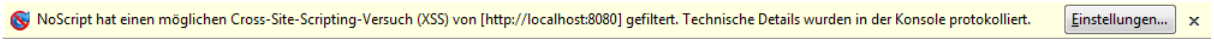

Abbildung 5–14: NoScript: Meldung – XSS-Versuch

Dieser kann den geblockten Request dann dennoch zulassen, falls dies eine gültige Anfrage war. Außerdem verhindert „NoScript“ das Laden von Dateien aus entfernten JAR-Archiven. Auch dies dient dem Schutz vor XSS-Angriffen. Bei beiden Kategorien lassen sich durch das Definieren regulärer Ausdrücke URLs bestimmen, die von der Filterung ausgenommen werden sollen. Die Filterung kann man durch Eingabe einer Beispieladresse überprüfen. Ist die URL rot, wird der Filter angewendet, ist sie schwarz, wird die Filterung umgangen.

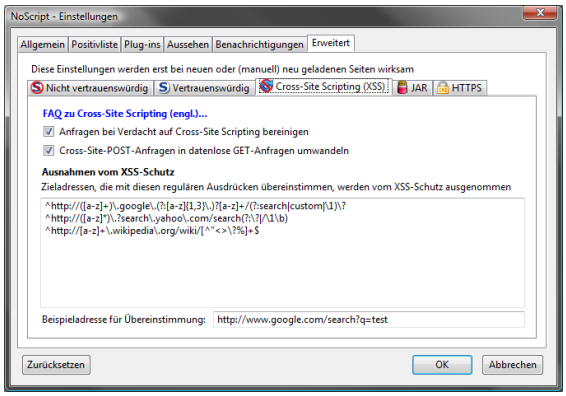

Abbildung 5–15: NoScript: Erweitert – XSS

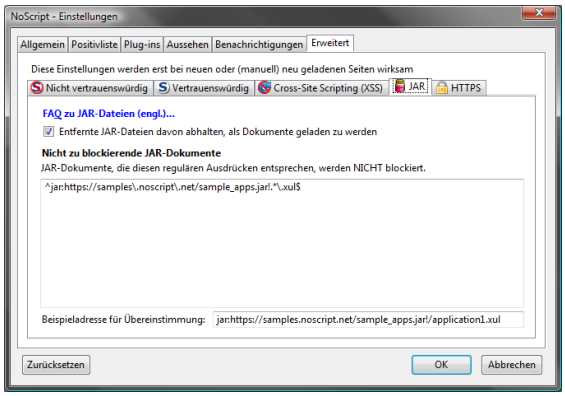

Abbildung 5–16: NoScript: Erweitert – JAR

In der letzten Kategorie „HTTPS“ kann man über die Unterkategorie „Verhalten“ festlegen, ob aktive Inhalte über eine sichere Verbindung übertragen werden müssen. Es gibt hierbei zwei Ausnahmelisten. Die erste beinhaltet Seiten, für die immer eine sichere Verbindung nötig ist, egal welche Inhalte sie anfordert. Die zweite Liste ist für Ausnahmeregelungen, bei denen https für aktive Inhalte nicht erzwungen wird, auch wenn diese für andere Seiten über eine sichere Verbindung übertragen werden müssen. In der Unterkategorie „Cookies“ kann man ein „Automatisches sicheres Cookie-Management“ aktivieren. Ist dies angeschaltet, setzt „NoScript“ das Sicherheitsflag bei Cookies, die über eine https-Verbindung ankommen, wenn sie in der ersten Liste aufgeführt wurden. Wird nur ein Cookie über die https-Verbindung gesendet, wird das Sicherheitsflag auch gesetzt, falls die Seite nicht in der Liste ist. In der zweiten Liste kann man Ausnahmen definieren, für die die Sicherheitsflags trotz https-Verbindung nicht gesetzt werden sollen.

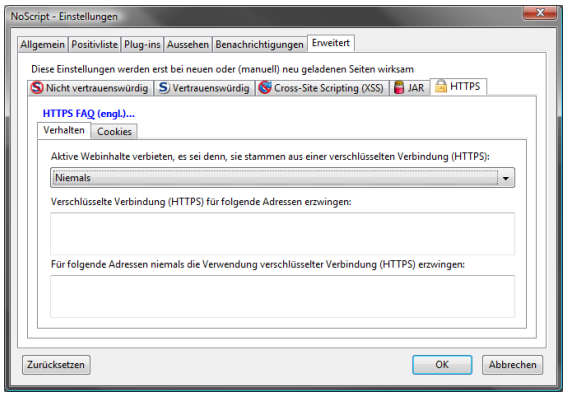

Abbildung 5–17: NoScript: Erweitert – HTTPS – Verhalten

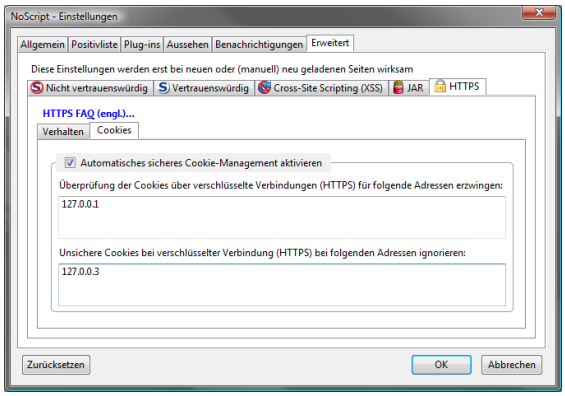

Abbildung 5–18: NoScript: Erweitert – HTTPS – Cookies

6 Fazit

Die Angriffstechniken der Online-Kriminellen werden immer ausgefeilter. Mittlerweile finden die Attacken nicht nur auf Webanwendungen statt, die Fehler in Authentifizierungsmechanismen beinhalten, sondern auch auf solchen, die an sich stark abgesichert sind. Wenn es Angreifern möglich ist, authentifizierte Verbindungen mitzubenutzen, da dies vom Browser begünstigt wird, ist es für die Anwendungen schwer, zwischen gewollter und gefälschter Anfrage zu unterscheiden. Zwar überlegen sich die Entwickler immer wieder neue, zusätzliche Sicherheitsmaßnahmen, jedoch werden meist wieder Wege gefunden, die auch diese Vorkehrungen täuschen oder aushebeln. Die Programmierer sind im Implementieren von Sicherheitsmechanismen weitestgehend auf die Serverseite eingeschränkt, wohingegen Angreifer Lücken server- und clientseitig ausnutzen können. Dies stellt ein Ungleichgewicht dar, das es auszugleichen gilt.

Deshalb ist es wichtig, dass auch clientseitig Sicherheit implementiert wird. Hierzu müssen die Browserhersteller in die Pflicht genommen werden. Sie sind dafür verantwortlich, dass Daten und Anwendungen zusätzlich geschützt werden. Dazu müssen die einzelnen Daten im Browser voneinander getrennt werden, so dass beispielsweise Cookies, die für eine Verbindung genutzt werden, nicht von einer anderen verwendet werden können. Außerdem muss der Benutzer vor Täuschungsversuchen bewahrt werden.

Leider kann man erkennen, dass die Browser die Sicherheit von Webanwendungen und die der Benutzerdaten kaum unterstützen, was zur Folge hat, dass selbige weder gewährleistet noch verbessert wird. Ganz im Gegenteil, sie verschlechtern den Schutz von Daten sogar. Man muss nur an den gemeinsam genutzten Cookiespeicher oder an eingebundene Applets aus anderen Domänen denken. Dies sind aber nicht die einzigen Mängel, die sich Browserhersteller zuschreiben lassen müssen. Eine besondere Gefahr rührt daher, dass die Browser Daten aus anderen Domänen automatisch laden. Dies hat zum einen zur Folge, dass Angriffe, die autorisierte Verbindungen mitbenutzen, wie CSRF oder auch GIFAR, überhaupt durchgeführt werden können. Zum anderen wird dadurch das Täuschen des Anwenders erleichtert, denn dieser assoziiert die angezeigten Inhalte normalerweise mit der aufgerufenen Adresse. Wenn nun eine schlecht gesicherte Webanwendung Frames verwendet, kann ein Angreifer diese dazu bringen, beliebige Inhalte in einem solchen Frame anzuzeigen. Der Browser weiß, woher er die Daten abruft, für den Nutzer ist dies auf den ersten Blick nicht nachvollziehbar. Auch wenn Browser ohne Benutzerbenachrichtigung das Senden von Formulardaten in eine andere Domäne zulassen, als in die, in der sich die aufgerufene Webseite befindet, ist dies ein schweres Vergehen. Gelingt es einem Hacker, in einer Webanwendung ein manipuliertes Formular anzuzeigen, kann er dadurch Daten klauen. Dabei wäre es dem Browser ein Leichtes, verdächtige Inhalte zu blockieren und den Benutzer zu benachrichtigen, denn dem Browser ist bekannt, woher er Daten lädt und wohin er Daten verschickt. Das Vorenthalten solcher Informationen bietet eine Basis für viele Angriffe.

Doch selbst wenn Einstellungen vorhanden sind, die die Sicherheit wesentlich erhöhen, werden diese vom Großteil der Nutzer nicht verwendet. Ein Grund hierfür liegt in einer schlechten Dokumentation dieser Einstellungen, was beim meist verbreiteten Browser Internet Explorer der Fall ist und dazu führt, dass Laien nichts damit anfangen können. Ein weiterer Grund besteht darin, dass viele nicht das technische Know-How besitzen, um die Sicherheitseinstellungen ihres Browsers an ihren Bedarf anzupassen. Andere meinen, dass sie geschützt sind, so lange sie keine zweifelhaften Seiten aufrufen. Es gibt auch Nutzer, die noch nie von Bedrohungen der Daten- und Anwendungssicherheit gehört

haben und sich deshalb keine Gedanken über ihre Sicherheit beim Surfen machen. Die Einstellungen beim IE sind zwar schlecht beschrieben, Punkte sammeln kann er bei diesen jedoch trotzdem. So kann man im Internet Explorer über das Zonenmodell unterschiedlichen Webseiten verschiedene Berechtigungen erteilen, was im Firefox bei einigen wichtigen Einstellungen nicht ohne Weiteres möglich ist. Ferner bilden beim IE die Einstellungsmöglichkeiten von verschiedenen Sicherheitsstufen über einen Regler für unkundige Benutzer eine Erleichterung zur Erhöhung der Sicherheit. Zudem können fortgeschrittene Benutzer ihre Einstellungen über „Stufe anpassen“ individuell festlegen. Leider sind die Sicherheitseinstellungen aber noch unzureichend und die Sicherheit einer Stufe deshalb meist trügerisch. Außerdem ist das Zonenmodell zu unflexibel, da nur vier Zonen unterstützt werden. Beschränkt man die Internetzone stark, müssen viele Seiten in die vertrauenswürdige Zone verschoben und dort alle Einstellungen gelockert werden, die die hinzugefügten Seiten benötigen. Dies kann zur Folge haben, dass auch Einstellungen aktiviert werden, die nicht für jede Webseite in der Zone erforderlich sind. Auch dies setzt die Sicherheit wieder herab. Hier punktet der Firefox mit seinen Ausnahmereglungen. Diese sind jedoch nicht für alle Einstellungen verfügbar und deshalb auch nicht optimal.

Oftmals liefern die Browser auch zu ungenaue Fehler- oder Warnmeldungen. Wird beispielsweise eine Meldung wie „Ausführung von Software wie ActiveX-Steuerelementen und Plugins zulassen?“ angezeigt, weiß der Benutzer nicht, welche ActiveX-Steuerelemente oder Plugins zugelassen werden sollen. Während einige eher harmlos sind, können andere sehr gefährlich sein. Ähnliche Fehlermeldungen gibt es auch in anderen Bereichen. Selbst ein Benutzer, der einiges an Wissen über gefährliche und nicht gefährliche Inhalte besitzt, kann bei solchen Meldungen aus Mangel an Informationen nicht immer die richtige Entscheidung treffen. Beiden Browsern gemein ist die schlechte Einschränkungsmöglichkeit von JavaScript. Da es in JavaScript viele nützliche aber auch einige schädliche Funktionen gibt, befindet sich der Benutzer im Zwiespalt, ob er nun JavaScript erlauben soll oder nicht. Verbietet er JavaScript, erhöht sich die Sicherheit gravierend. Allerdings sinkt auch die Benutzbarkeit von vielen Webseiten und manche sind schließlich gar nicht mehr benutzbar. Aktiviert der Anwender JavaScript, so verschlechtert sich die Sicherheit drastisch, wobei er aber die unterstützende oder benötigte Funktionalität vieler Web-Angebote weiterhin verwenden kann. Eine Abhilfe hierzu wurde bereits durch die beschriebenen Plugins aufgezeigt, über die sich bestimmte Methoden, Attribute oder Objekte sperren bzw. die Einstellungen für Seiten individuell festlegen lassen.

Wie zu erkennen ist, gibt es bei Browsern in Bezug auf Benutzerdaten- und Anwendungssicherheit noch viel zu verbessern. Da aber diese für die Browserhersteller bisher noch keinen hohen Stellenwert besitzt, wird die Umsetzung gewiss noch einige Zeit dauern. Deshalb ist man gut beraten, wenn man einen Browser verwendet, für den es Sicherheitsplugins gibt. Für den Internet Explorer existieren derzeit nur Browsererweiterungen, die den Browser selbst und das Betriebssystem gegen Angriffe schützen, nicht aber die Daten und Anwendungen. Deshalb rate ich zur Benutzung des Firefox mit dem NoScript-Plugin. Dennoch ist es auch mit Sicherheitsplugins wichtig, Webseiten gegenüber ein gesundes Misstrauen beizubehalten. Man sollte bei vertrauenswürdigen Seiten möglichst wenige Einstellungen aktivieren, nämlich nur diese, die wirklich gebraucht werden. Vor allem dort sollte man Vorsicht walten lassen, wo Anwender die Internet-Präsenz mitgestalten können, wie bei Online-Communities, Präsentationsseiten für bestimmte Inhalte, aber auch bei Foren. Denn hier ist es für Angreifer meist einfach, Manipulationen vorzunehmen. Aus diesem Grund sollte man speziell solche Seiten mit möglichst großen Einschränkungen belegen.

7 Ausblick

Die Daten- und Anwendungssicherheit besitzt bei den Browserherstellern eher einen untergeordneten Stellenwert. Deshalb sind in der nächsten Zeit auch keine großen Verbesserungen zu erwarten. Eine Änderung kann hier vielleicht durch das Cloud-Computing hervorgerufen werden, denn wenn kritische Anwendungen, die normalerweise lokal auf dem Rechner installiert waren, nun über das Internet betrieben werden, müssen weitere Sicherheitsmechanismen implementiert werden. So können beispielsweise bald Microsoft Office Dokumente über den Browser erstellt werden (Schüler, et al., 2008). In Folge dessen können die Browserhersteller durch den Einfluss der Betreiber von Cloud-Diensten dazu veranlasst werden, die Daten- und Anwendungssicherheit zu erhöhen.

Nachfolgend gebe ich Empfehlungen zur Verbesserung der Daten- und Anwendungssicherheit:

Als erstes sollten die Browserhersteller die Fehlermeldungen überarbeiten. Die Fehlermeldungen müssen kurz aber präzise sein. Werden sie aufgrund ihrer Genauigkeit sehr lang, sollte ein Detail-Button angezeigt werden, über den man weitere Informationen anzeigen lassen kann. Außerdem ist es sinnvoll, bei Aktionen, die die Webseite zwar einschränken aber nicht die Darstellung unterbinden, Meldungen in der Informationsleiste einzublenden statt Popups anzuzeigen. Die Informationsleiste muss fälschungssicher sein, wie dies in [Abbildung 3-3: IE mit Warnung (unsicher) – Seite 37] bis [Abbildung 3-6: IE ohne Warnung (sicher) – Seite 38] gezeigt wird. Für Maßnahmen, die die Darstellung einer Webseite verhindern, wie beispielsweise Zertifikatsfehler, sollten Meldungen anstatt der Webseite angezeigt werden, wie dies bereits schon oft der Fall ist. Popup-Meldungen sind störend und sollten nur in Ausnahmefällen verwendet werden, wenn die Meldung nicht in eine der vorher erwähnten Kategorien einzuordnen ist. Erscheinen zu häufig Hinweise in Popup-Fenstern, werden diese oft ungelesen bestätigt und so gefährliche Aktionen erlaubt.

Des Weiteren ist es sehr wichtig, dass die Browser bereits mit sicheren Standardeinstellungen ausgeliefert werden. Viele Benutzer installieren einen Browser und befassen sich nicht mit den Einstellungen. Aber gerade solche Benutzer müssen meist besonders geschützt werden, da sie auch leichter eine Fehleinschätzung vornehmen und somit einfacher durch Angreifer getäuscht werden können. Jegliche Sicherheitsmaßnahmen schlagen fehl, wenn sie vom Anwender nicht richtig benutzt werden. Deshalb sollte der Browser Überprüfungen vornehmen, die ohne Zutun des Anwenders Schutzmechanismen ausführen, wie beispielsweise kontrollieren, ob Benutzerdaten die Domänengrenze überschreiten. Weitere automatische Verfahren sollten im Standardverhalten selbstständig Entscheidungen treffen und erst über Einstellungen dazu veranlasst werden, beim Benutzer nachzufragen, was zu tun ist. Denn legt ein Browser einem unkundigen Benutzer zu viele Abfragefenster vor, ist es wahrscheinlicher, dass dieser ab und zu die falsche Entscheidung trifft. Stellt der Anwender eine Schutzmaßnahme so ein, dass er selbst gefragt wird, weiß er vermutlich auch, was er tut und wie er bei einer Meldung reagieren muss.

Einstellungen müssen flexibel definierbar sein. Damit ist gemeint, dass Berechtigungslisten angelegt und diesen dann gewissen Seiten zugewiesen werden können. Somit ist gewährleistet, dass gleiche Einstellungen nicht für unterschiedliche Webseiten wiederholt werden müssen und man trotzdem verschiedene Einstellungen für andere Gruppen festlegen kann. Einen sehr guten Ansatz bietet hier das Plugin „Controle de Scripts“, bei dem genau dieses Verhalten zu erkennen ist. Außerdem ist es wichtig, Feineinstellungen vornehmen zu können, mit denen nicht ein ganzer Satz von

Funktionalitäten auf einmal erlaubt wird. Dies wäre vor allem dann schlecht, wenn sichere und unsichere Funktionen gemischt sind. Ein gutes Beispiel hierfür bildet JavaScript. Kann man die Einstellungen auf einer niedrigen Ebene vornehmen, wie auf Ebene von Methoden, Attributen und Objekten, bringt dies ein Mehr an Sicherheit. Man kann dann getrost nützliche Funktionalitäten aktivieren, wohingegen man gefährliche nur vertrauenswürdigen Seiten vorbehält, die diese unbedingt benötigen um zu funktionieren.

Wichtig ist es auch, dass die bestehenden Schwachstellen im Browser selbst behoben werden. So dürfen Cookies nicht allen Fenstern bzw. Tabs zur Verfügung stehen, sondern nur dem, das es empfangen hat. Nur, wenn neue Fenster innerhalb derselben Domäne von einem Fenster aus geöffnet werden, darf der Browser die bestehenden Cookies auch für das neue freigeben. Der Browser muss außerdem sicherstellen, dass Anmeldedaten nur automatisch hinzugefügt werden, wenn der Benutzer den Login selbst vornimmt und nicht, wenn dies automatisiert geschehen soll. Außerdem darf der Browser nicht zulassen, dass Daten ohne die explizite Zustimmung des Anwenders über Domänengrenzen hinweg ausgetauscht werden. Hierfür muss der Browser den Anwender nicht nur bei Formular-Weiterleitungen warnen, sondern auch, wenn aktive Inhalte Daten über Domänengrenzen hinweg bewegen wollen. Es müssen geeignete Sicherheitsmechanismen eingeführt werden, die dies ermitteln können. Bei JavaScript wäre ein tainting-Mechanismus denkbar. Bei anderen Plugins wie Java oder Flash müssen Hersteller von Browser und Plugins zusammen eine Lösung erarbeiten. Auch auf der Ebene der Interaktionssicherheit müssen Verbesserungen vorgenommen werden. Ein Benutzer muss immer wissen, welche Aktion er tatsächlich auslöst. Bei einem Klick auf ein transparentes Objekt muss somit der Browser eine Warnmeldung ausgeben. Werden all diese Maßnahmen umgesetzt, ist die Daten- und Anwendungssicherheit auf einem sehr hohen Niveau.

Weitere Verbesserungen sind durch Absprachen zwischen Webanwendungsentwicklern und Browserherstellern möglich. Denn wenn eine Anwendung der Clientseite mitteilen kann, welche Privilegien sie benötigt, kann der Browser sehr strikte Einschränkungen vornehmen. Natürlich sollten Einstellungen des Benutzers Vorrang vor solchen automatischen Einstellungen haben. Teilt eine Webseite dem Browser beispielsweise mit, dass sie nur bestimmte JavaScript-Funktionen benötigt und ist der Zugriff auf Cookies über JavaScript nicht in der aufgeführten Berechtigungsliste, kann der Browser diese Funktion für die Seite sperren. Gelingt es nun einem Angreifer, der versucht das Cookie zu stehlen, JavaScript-Code in die Seite einzuschleusen, kann dies der Browser durch seine Einschränkung verhindern. Er könnte der Webseite sogar eine Rückmeldung liefern, dass ein von ihr nicht spezifiziertes Privileg genutzt werden sollte. So ist es den Anwendungsentwicklern möglich, clientseitige Angriffe nachzuvollziehen und gezielt nach Schwachstellen in ihrer Anwendung zu suchen. Es erscheint sinnvoll, hierfür einen Standard zu schaffen, über den Webanwendungen Clients mitteilen können, welche Berechtigungen sie benötigen. Denkbar wäre eine XML-Datei, die an einem definierten Ort zu finden ist.

Leider sind die aufgeführten Schritte nicht einfach zu verwirklichen, besonders, wenn sie verschiedene Interessengruppen und Unternehmen betreffen. Deswegen wird noch etwas Zeit verstreichen, bis solche oder vergleichbare Maßnahmen umgesetzt werden.

Quellenverzeichnis

Aharonovsky, Guy. 2008. Camera Clickjacking - The Game. [Online] 08. Oktober 2008. [Zitat vom: 25. November 2008.] http://guya.net/security/clickjacking/game.html.

—. 2008. Malicious camera spying using ClickJacking. [Online] 07. Oktober 2008. [Zitat vom: 25. November 2008.] http://blog.guya.net/.

Bachfeld, Daniel. 2008. Clickjacking: Jeder Klick im Browser kann der falsche sein. [Online] Heise Security, 08. Oktober 2008. [Zitat vom: 28. Oktober 2008.] http://www.heise.de/security/Clickjacking-Jeder-Klick-im-Browser-kann-der-falsche-sein--/news/meldung/117055.

—. 2008. Einfallstor Browser. [Online] Heise Security, 12. September 2008. [Zitat vom: 27. Oktober 2008.] http://www.heise.de/security/Einfallstor-Browser--/artikel/115254/0.

—. 2008. Schwachstelle in Banken-Site ermöglichte unautorisierte Überweisung. [Online] Heise Security, 01. Oktober 2008. [Zitat vom: 17. November 2008.] http://www.heise.de/security/Schwachstelle-in-Banken-Site-ermoeglichte-unautorisierte-Ueberweisung-Update--/news/meldung/116767.

—. 2008. Spionagefotos, mal andersherum. [Online] Heise Security, 01. August 2008. [Zitat vom: 30. Oktober 2008.] http://www.heise.de/security/Spionagefotos-mal-andersherum-2-Update--/news/meldung/113684.

Born, Günter. 2004. *Windows XP Home Tricks.* 2. Auflage. München : Markt und Technik, 2004. 978-3-8272-6837-2.

Computerwoche. 2007. IT-Sicherheit: Die zehn gefährlichsten Schwachstellen in Web-Anwendungen. [Online] IDG Business Media GmbH, 22. Oktober 2007. [Zitat vom: 29. Oktober 2008.] http://www.computerwoche.de/knowledge_center/security/582260/index.html.

Diercks, Jürgen. 2008. Micro Focus will große Anwendungen in Clouds auslagern. [Online] iX, 15. Dezember 2008. [Zitat vom: 15. Dezember 2008.] http://www.heise.de/ix/Micro-Focus-will-grosse-Anwendungen-in-Clouds-auslagern--/news/meldung/120439.

Gagnon, Réal. 1997. Real's HowTo. [Online] 1997. [Zitat vom: 20. November 2008.] http://www.rgagnon.com/howto.html.

Google Inc. 2007. Google Safe Browsing for Firefox. [Online] 2007. [Zitat vom: 09. Januar 2009.] http://www.google.com/tools/firefox/safebrowsing/.

Hansen, Robert. 2008. Clickjacking Details. [Online] SecTheory, 07. Oktober 2008. [Zitat vom: 25. November 2008.] http://ha.ckers.org/blog/20081007/clickjacking-details/.

Hansen, Robert und Grossman, Jeremiah. 2008. Clickjacking. [Online] SecTheory, 12. September 2008. [Zitat vom: 25. November 2008.] http://www.sectheory.com/clickjacking.htm.

Heise Security. 2006. Sicherheitseinstellungen des Internet Explorer 7 anpassen. [Online] 11. Juli 2006. [Zitat vom: 10. Dezember 2008.] http://www.heise.de/security/dienste/browsercheck/anpassen/ie70/default.shtml.

Huseby, Sverre H. 2004. *Sicherheitsrisiko Web-Anwendung.* 1. Auflage. 69115 Heidelberg : dpunkt.verlag GmbH, 2004. 978-3-89864-259-0.

Janowicz, Krzysztof. 2006. *Sicherheit im Internet.* 2. Auflage. Köln : O'Reilly, 2006. 978-3-89721-425-5.

Maone, Giorgio. 2008. Hello ClearClick, Goodbye Clickjacking! [Online] hackademix.net, 08. Oktober 2008. [Zitat vom: 26. November 2008.] http://hackademix.net/2008/10/08/hello-clearclick-goodbye-clickjacking/.

Mastracci, Matthew. 2008. Clickjacking demo. [Online] 27. Oktober 2008. [Zitat vom: 26. November 2008.] http://grack.com/record/.

Microsoft Corporation. 2008. Internet Explorer Gallery. [Online] 2008. [Zitat vom: 27. Oktober 2008.] http://ieaddons.com/en/.

Mozilla Foundation. 2008. Controle de Scripts 1.0.3. [Online] 29. Mai 2008. [Zitat vom: 10. Januar 2009.] https://addons.mozilla.org/de/firefox/addon/1154.

—. 2008. Firefox-Add-ons. [Online] 27. Oktober 2008. [Zitat vom: 27. Oktober 2008.] https://addons.mozilla.org/de/firefox/.

—. 2009. NoScript 1.8.9.2. [Online] 07. Januar 2009. [Zitat vom: 10. Januar 2009.] https://addons.mozilla.org/de/firefox/addon/722.

MozillaZine. 2008. About:config entries. [Online] 26. Dezember 2008. [Zitat vom: 07. Januar 2009.] http://kb.mozillazine.org/About:config_Entries.

—. 2006. Category:Preferences. [Online] 05. Februar 2006. [Zitat vom: 10. Januar 2009.] http://kb.mozillazine.org/Category:Preferences.

—. 2007. Security Policies. [Online] 01. Januar 2007. [Zitat vom: 13. November 2008.] http://kb.mozillazine.org/Security_Policies.

Nasarek, Marcus. 2007. *Windows Vista Security.* 1. Auflage. 50670 Köln : O'Reilly, 2007. 978-3-89721-466-8.

Netzwelt. 2005. Internet Explorer Teil 3: Wo Profis ins Schwitzen kommen. [Online] 24. Januar 2005. [Zitat vom: 21. Dezember 2008.] http://www.netzwelt.de/news/69490-internet-explorer-teil-3-wo.html.

O'Callahan, Robert. 2002. Bug 154957 - iframe content background defaults to transparent. [Online] 30. Juni 2002. [Zitat vom: 20. November 2008.] https://bugzilla.mozilla.org/show_bug.cgi?id=154957#c6.

OWASP Foundation. 2008. OWASP Main Page. [Online] 28. Oktober 2008. [Zitat vom: 27. Oktober 2008.] http://www.owasp.org/index.php/Main_Page.

Petkov, Petko D. 2007. Java JAR Attacks and Features. [Online] Gnucitizen, 10. November 2007. [Zitat vom: 20. November 2008.] http://www.gnucitizen.org/blog/java-jar-attacks-and-features/.

Ruderman, Jesse. 2006. Configurable Security Policies. [Online] Mozilla Foundation, 08. April 2006. [Zitat vom: 21. November 2008.] http://www.mozilla.org/projects/security/components/ConfigPolicy.html.

Rütten, Christiane. 2008. Neue Bedrohung durch "Clickjacking"? [Online] Heise Security, 18. September 2008. [Zitat vom: 28. Oktober 2008.] http://www.heise.de/security/Neue-Bedrohung-durch-Clickjacking--/news/meldung/116170.

Rütten, Christiane und Glemser, Tobias. 2007. Sicherheit von Webanwendungen. [Online] Heise Security, 25. Januar 2007. [Zitat vom: 29. Oktober 2008.] http://www.heise.de/security/Sicherheit-von-Webanwendungen--/artikel/84149/0.

Schüler, Hans-Peter und Briegleb, Volker. 2008. PDC: MS Office auch übers Web. [Online] Heise Online, 29. Oktober 2008. [Zitat vom: 4. Januar 2009.] http://www.heise.de/newsticker/PDC-MS-Office-auch-uebers-Web--/meldung/118080.

Sun Microsystems, Inc. 2008. How Cookie Support Works in Java Plug-in. [Online] 23. Juni 2008. [Zitat vom: 20. November 2008.] http://java.sun.com/products/plugin/1.2/docs/cookie.html.

—. 2008. JavaDoc JSObject. [Online] 06. Oktober 2008. [Zitat vom: 25. November 2008.] http://java.sun.com/javase/6/webnotes/6u10/plugin2/liveconnect/jsobject-javadoc/netscape/javascript/JSObject.html.

Wilkens, Andreas. 2008. Microsoft: Beim Cloud Computing wird es nur wenige Konkurrenten geben. [Online] Heise Online, 10. November 2008. [Zitat vom: 10. November 2008.] http://www.heise.de/newsticker/Microsoft-Beim-Cloud-Computing-wird-es-nur-wenige-Konkurrenten-geben--/meldung/118663.

Zocholl, Michaela. 2005. *Windows XP Professional.* München : Addison-Wesley, 2005. 978-3-8273-2237-1.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelor-Thesis selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum

Unterschrift