Diplomarbeit

Verbesserung des Datenschutzes bei Webanwendungen

Maria Klemm

9. Juni 2011

Technische Universität Dresden
Fakultät Informatik
Institut für Systemarchitektur
Professur für Betriebssysteme

Betreuender Hochschullehrer: Prof. Dr. rer. nat. Hermann Härtig
Betreuender Mitarbeiter: Dipl.-Inf. Carsten Weinhold

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig erstellt und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Dresden, den 9. Juni 2011

Maria Klemm

Danksagung

Ich bedanke mich an dieser Stelle bei Professor Hermann Härtig dafür, dass er mir die Möglichkeit gegeben hat, meine Diplomarbeit am Lehrstuhl für Betriebssysteme zu schreiben.
Ein ganz besonders großes Dankeschön geht an meinen Betreuer Carsten Weinhold und seine Kollegen Michael Roitzsch und Björn Döbel. Sie hatten stets ein offenes Ohr für meine Fragen und Probleme und haben mit ihren kritischen Anmerkungen und wertvollen Hinweisen immer wieder dafür gesorgt, dass ich meine Motivation nicht verliere.

Inhaltsverzeichnis

1 Einleitung **1**

2 Grundlagen **3**
2.1 Webanwendung 3
2.2 Angriffskonzepte 5
2.2.1 Angriff auf den Datenverkehr 5
2.2.2 Angriff im Namen des Nutzers 6
2.2.3 Angriff durch den Dienstanbieter 7

3 Verwandte Arbeiten **9**
3.1 Kommunikation zwischen Webanwendungen 9
3.2 Verbesserte Funktionsweise 11
3.3 Schutz vertraulicher Daten 12
3.4 Unzuverlässige Clients 14
3.5 Speicherung auf unzuverlässigen Servern 16

4 Entwurf **19**
4.1 Verbesserter Datenschutz 20
4.2 Eignung von Webanwendungen 21
4.3 Instrumentierung der Webanwendung 21
4.3.1 Vertraulichkeit 21
4.3.2 Integrität 22
4.4 Anpassungen im Browser 23
4.4.1 Verschlüsselung 23
4.4.2 Authentifizierung 24
4.5 Ablauf einer instrumentierten Webanwendung 25
4.5.1 Registrierung 25
4.5.2 Ausführung 25

5 Implementierung **29**
5.1 Persistenter Key-Value-Store 29
5.2 Browsererweiterung 30
5.3 Instrumentierung der ToDo-Liste 32
5.4 Authentifizierung der ToDo-Liste 34

6 Bewertung **39**
6.1 Erreichte Verbesserung des Datenschutzes 39
6.1.1 ToDo-Liste aus der Sicht des Nutzers und des Dienstanbieters . 39
6.1.2 Ausführung ohne Berechtigung 41
6.1.3 Manipuliertes JavaScript 41
6.1.4 Entschlüsselungsversuch einer anderen Webanwendung 42

6.2 Aufwand 43
6.3 Messungen 44
6.3.1 Dauer einer Verschlüsselung und einer Entschlüsselung 44
6.3.2 Ladezeit der ToDo-Liste 46

7 Ausblick und Zusammenfassung **49**
7.1 Ausblick 49
7.2 Zusammenfassung 49

Abbildungsverzeichnis **51**

Tabellenverzeichnis **53**

Glossar **55**

Literaturverzeichnis **57**

1 Einleitung

Webanwendungen erfreuen sich immer größerer Beliebtheit. Mittlerweile verwaltet ein durchschnittlicher Nutzer nicht nur seine Emails online, er verabredet Termine über Webanwendungen wie zum Beispiel Doodle, erledigt seine Bankgeschäfte online, bestellt die unterschiedlichsten Waren und legt sich ein Adressbuch oder einen Kalender an, um seine Kontaktdaten oder Termine jederzeit und überall zur Verfügung zu haben, ohne sie bei sich tragen zu müssen. Viele Nutzer dieser Webanwendungen gehen davon aus, dass der Dienstanbieter nicht nur für die Funktionalität der Webanwendung sorgt, sondern auch für einen verantwortungsvollen Umgang mit den Daten der Nutzer.

Die meisten Nutzer konnten mittlerweile dafür sensibilisiert werden, dass man für die Übertragung vertraulicher Daten wie Kontonummern und Kreditkarteninformationen eine SSL-Verbindung nutzen sollte. Wie verantwortungsvoll jedoch auf der Server-Seite mit den Daten des Nutzers, wie etwa Kalendereinträgen oder Adressen, umgegangen wird, liegt immer noch in der Hand des Dienstanbieters. Für ihn gibt es auf den ersten Blick zwei unterschiedliche Strategien für den Umgang mit privaten Daten, die der Nutzer ihm anvertraut hat. Er kann den größtmöglichen Nutzen daraus ziehen, indem er alle möglichen Informationen über den Nutzer sammelt und diese zu Werbezwecken verwendet oder weiterverkauft. Die gegenläufige Strategie für den Dienstanbieter wäre, alle denkbaren Vorkehrungen zu treffen, so dass die Daten des Nutzers ausschließlich der Webanwendung zur Verfügung stehen. Dies würde beinhalten, dass ein möglicher Angreifer nicht in der Lage ist, die Daten zu erhalten oder zu verändern.

Dass die Privatsphäre der Nutzer von den Dienstanbietern selten respektiert wird, zeigt eine Studie [1], die untersucht hat, wie mit den sensiblen Daten der Nutzer in Webanwendungen umgegangen wird. Dabei wurde festgestellt, dass es unter den 50.000 meistbesuchten Webseiten, laut Alexa Rangliste, 485 Webseiten gibt, die sich unerlaubt Zugriff auf die privaten Daten der Nutzer verschaffen. Alexa ist ein eine Firma, die anhand von Besucherzahlen eine Rangliste der weltweit am häufigsten besuchten Webseiten ermittelt [2]. In Abschnitt 4.3.1 wird beschrieben, in welcher Form diese 485 Webseiten das Vertrauen der Nutzer missbrauchen.

Ein Nutzer dem bewusst ist, wie viel Vertrauen er seinem Dienstanbieter entgegenbringt, hat im Moment nur die Wahl, seinem Dienstanbieter zu vertrauen oder auf die Webanwendung zu verzichten. Es ist jedoch möglich, die Daten während ihrer Übertragung an den Dienstanbieter vor unberechtigten Zugriffen zu schützen. Mit einer SSL-Verbindung können beispielsweise die Nutzer von Onlinebankgeschäften verhindern, dass ihre Daten während der Übertragung ausspioniert oder manipuliert werden. Durch das SSL-Schlosssymbol können sich die Nutzer ohne viel Aufwand davon überzeugen, dass ihre Daten über eine verschlüsselte Verbindung übertragen werden. Nützlich wäre ein ähnlicher Mechanismus mit dem sich die Nutzer vergewissern können, dass ihre Daten geschützt sind, wenn sie diese dem Dienstanbieter einer Webanwendung anvertraut haben. Würde dem Nut-

zer ein entsprechendes optisches Signal anzeigen, dass er eine Webanwendung benutzt, die alle Daten auf der Server-Seite nur verschlüsselt ablegt, könnte auch ein datenschutzbewusster Nutzer diese Webanwendung ausführen, weil das notwendige Vertrauen gegenüber dem Dienstanbieter reduziert wurde. Der Nutzer sollte für einen verbesserten Datenschutz keine zusätzlichen Kenntnisse oder Fähigkeiten benötigen, sondern lediglich einen Browser der ihm anzeigt, ob gerade eine der bisher üblichen Webanwendungen ausgeführt wird, oder eine mit verbessertem Datenschutz.

Ziel dieser Arbeit ist es, einen solchen Mechanismus zu realisieren. Dafür wird ein Browser so modifiziert, dass er die Daten des Nutzers kryptographisch absichern kann. Zudem wird eine Webanwendung so instrumentiert, dass sie vor dem Senden und nach dem Empfangen von Daten diese zusätzliche kryptographische Absicherungsfunktionaliät im Browser aufruft. Das notwendige Vertrauen, das der Nutzer dem Dienstanbieter einer Webanwendung entgegenbringen muss, wird dadurch verringert.

2 Grundlagen

Dieses Kapitel dient dazu, eine Übersicht über die Themen zu geben, die mit der Aufgabenstellung verbunden sind. Zunächst wird beschrieben was passiert, wenn ein Nutzer eine Webanwendung ausführt, inklusive der beteiligten Akteure und Kommunikationswege. Danach werden mögliche Angriffe auf die Privatsphäre des Nutzers erläutert.

2.1 Webanwendung

Die im Folgenden beschriebenen Akteure und Interaktionen sind in der Abbildung 2.1 dargestellt.

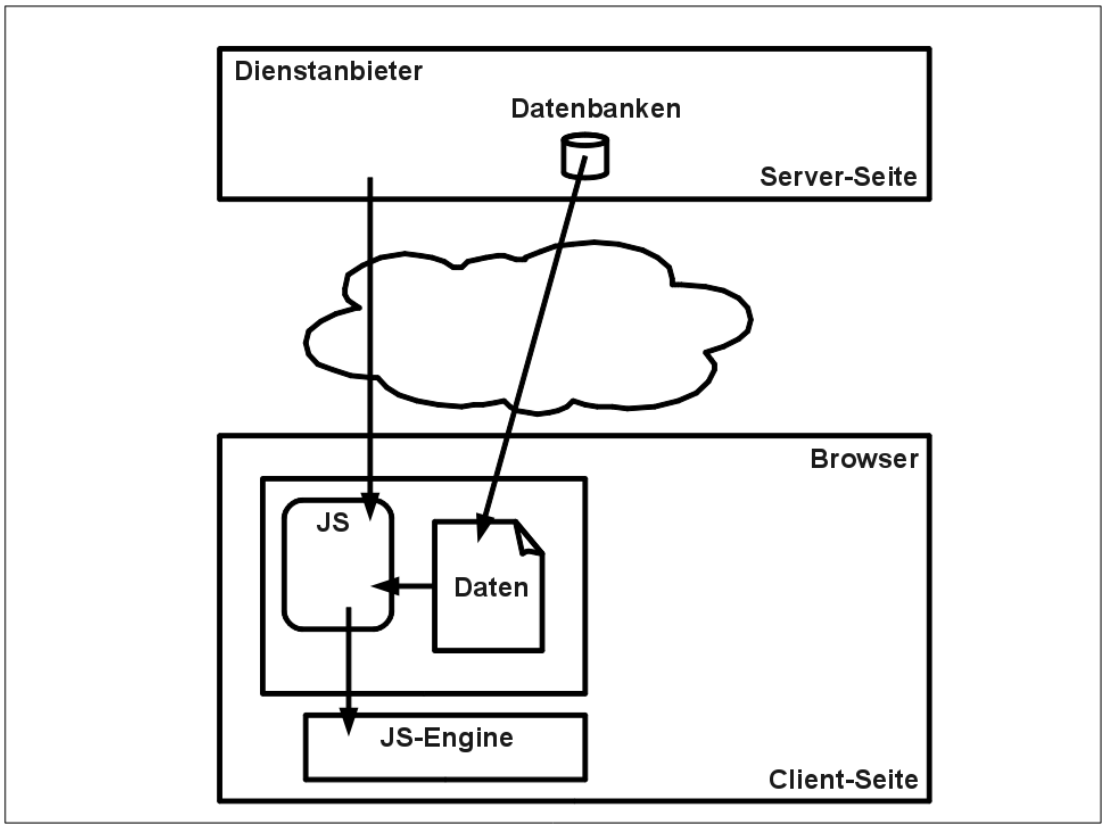

Abbildung 2.1: Modell einer Webanwendung

Der erste Schritt, den ein Nutzer ausführt, wenn er eine Webanwendung verwenden möchte, ist die Eingabe der entsprechenden URL in seinen Browser. Der Nutzer und sein Browser werden im Folgenden als die Client-Seite der Webanwendung bezeichnet, während der Dienstanbieter mit dem Webserver und den dazugehörigen

Datenbanken die Server-Seite darstellen. Die Client-Seite und die Server-Seite der Webanwendung kommunizieren miteinander in beiden Richtungen über das Internet (in Abbildung 2.1 als Wolke skizziert). Der Client sendet HTTP-Anfragen (Requests) an den Server, die jener mit HTTP-Antworten (Responses) erwidert. Je nach Art der Anfrage werden Daten angefordert oder übermittelt.

Durch die Eingabe der URL, erhält der Browser den Namen des Servers und löst ihn mit Hilfe von DNS zu einer konkreten IP-Adresse auf. Dorthin wird vom Browser eine HTTP-GET-Anfrage gesendet, durch die der Inhalt der Webseite angefordert wird, meistens ein HTML-Dokument. Wenn die Adresse gültig und das Dokument vorhanden ist, wird der Inhalt der Seite in den Browser herunter geladen. Der Browser interpretiert das HTML-Dokument und zeigt es dem Nutzer an.

Die Webseiten können JavaScript (JS) beinhalten, das auf der Client-Seite ausgeführt wird. Webanwendungen können so ihren Quellcode in einen server-seitigen und einen client-seitigen Part aufteilen und damit einen Teil der Last auf den Client übertragen. Solche Webanwendungen werden auch als Web-2.0-Anwendungen bezeichnet. Wenn in der Webanwendung JavaScript-Dateien vorhanden sind, werden sie über das Internet in zusätzlichen HTTP-GET-Anfragen angefordert und heruntergeladen. Die JavaScript-Engine des Browsers führt diese danach aus und dem Nutzer wird die Webanwendung angezeigt.

Das JavaScript wird dabei vom Browser durch die Same-Origin-Policy beschränkt. Sie verhindert, das eine JavaScript-Datei welche aus einer Quelle stammt die Inhalte aus anderen Quellen lesen, verändern oder ausführen kann. Der Wirkungskreis von JavaScript ist somit auf die Dokumente aus derselben Quelle begrenzt.

Sobald der Nutzer Daten in die Webanwendung eingibt, werden diese in HTTP-Anfragen an den Server übertragen. Mit den Eingaben des Nutzers können weitere Daten vom Server angefordert werden, die dann ebenfalls über das Internet in HTTP-Antworten an den Browser gesendet werden.

Die Webseiten können auch personalisierte Anwendungen, wie zum Beispiel einen Kalender oder ein Adressbuch für verschiedene Nutzer bereitstellen. Damit jeder Nutzer nur die zu ihm gehörenden Daten verwendet, müssen sich die Nutzer gegenüber dem Server mit einem Benutzernamen und einem Passwort identifizieren. Danach werden die Daten des Nutzers angefordert. Er kann Termine oder Adressen eingeben, woraufhin das JavaScript weitere HTTP-POST- oder HTTP-PUT-Anfragen auslöst, welche die Daten über das Internet an den Server übertragen.

Die übermittelten Daten werden auf der Server-Seite in Datenbanken gespeichert. Sobald ein Nutzer seine Webanwendung erneut aufruft, werden seine spezifischen Daten von seinem Browser angefordert und empfangen.

Solange der Nutzer mit dieser Webanwendung arbeitet, können neue Daten erstellt werden und bereits bestehende Daten verändert oder gelöscht werden. Der Nutzer beendet die Webanwendung, indem er sich ausloggt. Die Daten, die der Nutzer in der Webanwendung verwendet, befinden sich während der Ausführung der Webanwendung bei ihm im Browser. Sie können Teil von Übertragungen sein, die in beide Richtungen zwischen der Server-Seite und der Client-Seite über das Internet stattfinden und sie sind in den Datenbanken auf der Server-Seite vorhanden. An jedem dieser Orte können diese persönlichen, schützenswerten Daten des

Nutzers Angriffen ausgesetzt sein. Die Anforderungen an die Sicherheit von schützenswerten Daten werden unter dem Begriff Datenschutz zusammengefasst.

Datenschutz bezeichnet die Bewahrung persönlicher Daten vor Diebstahl, Missbrauch, Veränderung und Löschung. Im Allgemeinen verbindet man damit die Schutzziele Vertraulichkeit, Integrität und Verfügbarkeit. Wenn ein Nutzer seine vertraulich zu behandelnden, persönlichen Daten einer Webanwendung und somit dem Dienstanbieter anvertraut, geht er davon aus, dass dieser sie nicht an Dritte weitergibt und dass ihre Integrität nicht verletzt wird. Das heißt, er findet sie in dem Zustand wieder vor, in dem er sie eingegeben hat. Außerdem geht der Nutzer davon aus, dass er jederzeit auf alle seine Daten zugreifen kann.

Der folgende Abschnitt beschreibt die verschiedenen Angriffsmöglichkeiten auf die persönlichen Daten des Nutzers.

2.2 Angriffskonzepte

Ein Angreifer hat mehrere Möglichkeiten sich unberechtigten Zugriff auf diese Daten zu verschaffen. Gelingt ihm das, kann er sie auswerten, manipulieren oder zerstören. In welchen Situationen die Daten des Nutzers nicht ausreichend geschützt sind, zeigen die folgenden Szenarien.

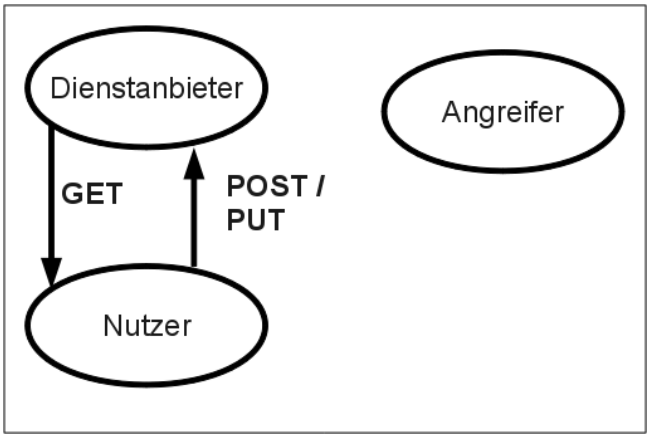

Abbildung 2.2: Angriffsmodell

Die drei Akteure des Modells sind der Nutzer mit seinem Browser auf der Client-Seite, der Dienstanbieter auf der Server-Seite mit dem Webserver und den Datenbanken und der Angreifer. Zwischen dem Client und dem Server werden die Daten mit GET-, POST- und PUT-Anfragen übertragen (siehe Abbildung 2.2). Das Ziel des Angreifers ist es, Zugriff auf die Daten des Nutzers zu erhalten.

2.2.1 Angriff auf den Datenverkehr

Ein Angreifer der den Datenverkehr in einer ungesicherten HTTP-Verbindung mitschneiden kann, ist in der Lage festzustellen, welche Daten zwischen dem Client und dem Server ausgetauscht werden (siehe Abbildung 2.3). Dass dies ohne viel Aufwand möglich ist, zeigt ein aktuelles Add-On für den Mozilla Firefox Browser namens Firesheep [3].

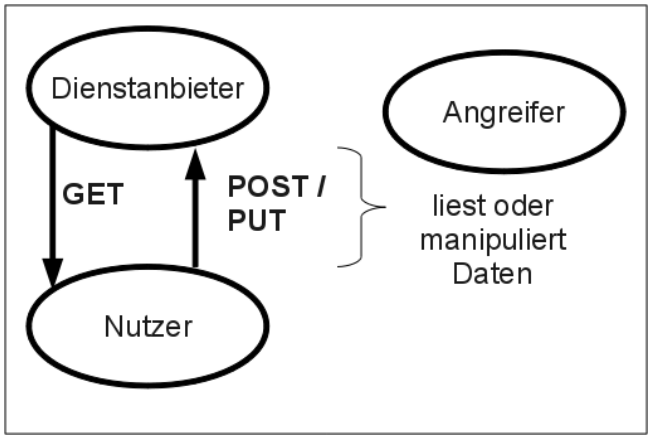

Abbildung 2.3: Angriff auf die Datenübertragung

Dadurch erfährt der Angreifer, welche Informationen der Nutzer auf dem Server hinterlegt hat. Ein passiver Angreifer kann so vertrauliche Daten des Nutzers erhalten. Ein aktiver Angreifer hat die Möglichkeit die Daten zu manipulieren und somit ihre Integrität zu verletzen. Zudem kann er in den Datenstrom eingreifen und das JavaScript der Webanwendung verändern, ersetzen oder andere JavaScript-Elemente hinzufügen. Dadurch kann sich die Funktionsweise der Webanwendung grundlegend ändern.

2.2.2 Angriff im Namen des Nutzers

Wenn der Angreifer sich als Nutzer ausgibt, nachdem er zum Beispiel den Loginvorgang mitprotokolliert hat oder Zugriff auf den Rechner des Nutzers bekommen hat, kann er wie der Nutzer auf die Webanwendung zugreifen (siehe Abbildung 2.4). Er sieht die privaten Daten des Nutzers und kann sie in seinem Namen beliebig verändern.

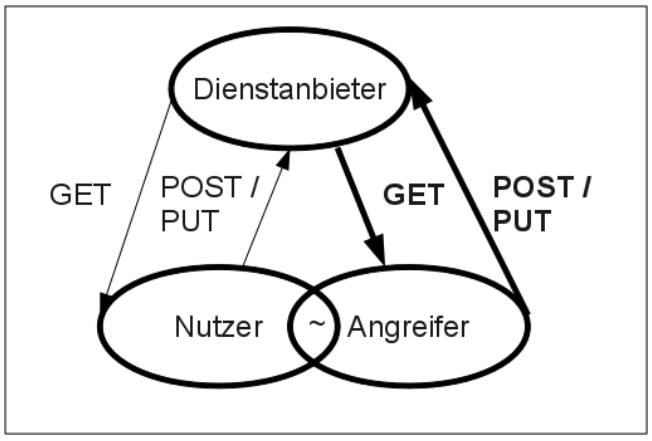

Abbildung 2.4: Angriff im Namen des Nutzers

2.2.3 Angriff durch den Dienstanbieter

Weil der Dienstanbieter die Daten für den Nutzer verwaltet, hat er auch Zugriff auf ihren Inhalt (siehe Abbildung 2.5).

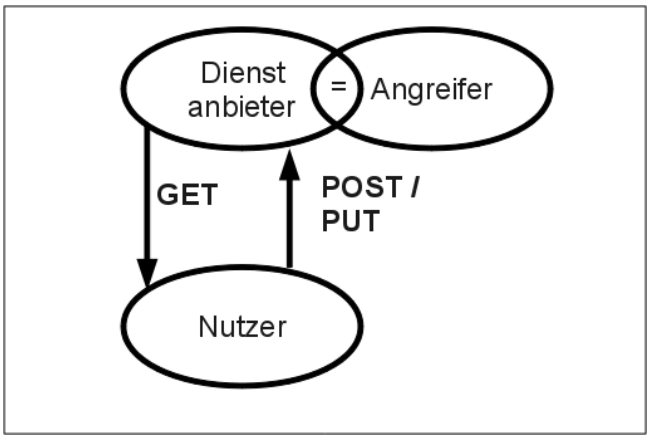

Abbildung 2.5: Angriff durch den Dienstanbieter

Dadurch kann der Dienstanbieter zum Angreifer werden, wenn er das Vertrauen ausnutzt, das der Nutzer ihm entgegenbringt. Dies geschieht wenn Informationen über den Nutzer nicht nur für die vereinbarte Dienstleistung verwendet werden, sondern zum Beispiel weiterverkauft oder für Werbezwecke genutzt werden.

Für die Angriffe auf den Datenverkehr und im Namen des Nutzers existiert bereits eine wirkungsvolle Schutzmaßnahme. Die HTTP-Verbindung zwischen dem Client und dem Server wird mit Hilfe einer SSL-Verschlüsselung geschützt, so dass die Daten manipulations- und abhörsicher über das Internet übertragen werden. Eine derart geschützte Verbindung wird als HTTPS-Verbindung bezeichnet. Ein Angreifer hat dadurch nicht mehr die Möglichkeit, das JavaScript oder Daten, die übertragen werden unbemerkt zu manipulieren. Ebenso wenig kann der Benutzername und das Passwort eines Nutzer bei der Übertragung über eine HTTPS-Verbindung ausgespäht werden. Gegen Angriffe auf den Rechner des Nutzers, beispielsweise durch einen Keylogger, der Benutzernamen und Passwörter mitprotokolliert, kann eine sichere Verbindung jedoch nichts ausrichten. Der Einsatz von HTTPS-Verbindungen bei Webanwendungen, die mit sensiblen Daten umgehen, ist weit verbreitet und schützt eine Webanwendung vor Angriffen wie sie in den Abschnitten 2.2.1 und 2.2.2 beschrieben wurden. Die Realisierung einer HTTPS-Verbindung liegt nicht im Fokus dieser Arbeit, sie wird aber als gegeben vorausgesetzt.

Für eine Webanwendung, die durch eine sichere HTTP-Verbindung vor Angriffen auf den Datenverkehr und im Namen des Nutzers geschützt ist, verbleibt die Verletzlichkeit für Angriffe durch den Dienstanbieter. Ziel dieser Arbeit ist es, einen Mechanismus zu entwickeln, der den Nutzer auch vor solchen Angriffen schützt.

Das folgende Kapitel beschreibt verschiedene Veröffentlichungen, die sich zum Ziel gesetzt haben, Webanwendungen sicherer zu machen. Dabei werden Bedrohungen auf der Server- und der Client-Seite berücksichtigt, sowie die Funktions-

weise und die Kommunikationsmöglichkeiten der Webanwendungen und deren Umgang mit vertraulichen Daten.

3 Verwandte Arbeiten

Dieses Kapitel stellt verschiedene Verfahren vor, die mit der Aufgabenstellung dieser Arbeit in Verbindung stehen und verwandte Aspekte beleuchten, die zur Verbesserung von Webanwendungen beitragen.

3.1 Kommunikation zwischen Webanwendungen

In diesem Abschnitt werden zwei Verfahren vorgestellt, die sich mit Schwachstellen der Same-Origin-Policy auseinandersetzen und mit ihren Lösungsansätzen die Kommunikation zwischen Webanwendungen feingranularer beschränken können.

Wissenschaftler der Carleton Universität in Ottawa haben Angriffe wie Cross-Site-Scripting (XSS), Cross-Site-Request-Forgery (XSRF) und andere untersucht und ein Verfahren entwickelt, das die Durchführung dieser Angriffe verhindert. Ein solcher Angriff kann durchgeführt werden, wenn es möglich ist, Informationen zu beliebigen, potentiell bösartigen Webservern zu senden oder von ihnen zu empfangen. Das **Same Origin Mutual Approval** (SOMA) Verfahren verhindert solche uneingeschränkten Informationsübertragungen [4]. Betrachtet wird hier das Einfügen von Inhalten externer Webserver in eine Webseite. Dieses Verfahren ist eine Verschärfung der Same-Origin-Policy (SOP). Die SOP verhindert, dass Dateien oder JavaScript aus einer Quelle die Inhalte aus einer anderen Quelle lesen, verändern oder ausführen können. Sie verhindert jedoch nicht, das Inhalte von externen Webservern abgerufen oder an diese gesendet werden. Das SOMA-Verfahren hat das Ziel, die SOP zu verschärfen, so dass die Ausnutzung von Schwachstellen besser verhindert werden kann. Schwachstellen bestehen zum einen in einer uneingeschränkten ausgehenden Kommunikation von der Ursprungsseite zu einem beliebigen Webserver, zum anderen in der Ausführung eines XSS-Angriffs oder eines XSRF-Angriffs. Dadurch können Informationen, wie beispielsweise ein Cookie oder die Logindaten des Nutzers gestohlen werden.

Das SOMA-Verfahren erfordert, dass beide, die Ursprungsseite und die Seite von der Inhalt eingebettet werden soll, die Anfrage genehmigen müssen, bevor jeglicher Inhalt angefordert wird. Dadurch erhält der Betreiber der Seite eine bessere Kontrolle darüber, was von anderen Seiten in seine Seite eingefügt wird.

Realisiert wird die gegenseitige Zustimmung durch eine Manifestdatei auf dem Webserver der Ursprungsseite und eine Genehmigungsdatei auf dem Webserver der Seite, die Inhalte anbietet. In der Manifestdatei wird festgelegt, welchen Domains es erlaubt ist, der Ursprungsseite Inhalte zur Verfügung zu stellen. Die Genehmigungsdateien auf den Webservern der Inhaltsanbieter zeigen an, welchen Seiten es erlaubt ist, Inhalte von dieser Seite bei sich einzufügen. Die Überprüfung und Durchsetzung dieser Mechanismen wird vom Webbrowser durchgeführt.

Die Grenzen des SOMA-Verfahrens sehen folgendermaßen aus: Mit SOMA ist es möglich, externe Kommunikation und das Einfügen von externen Inhalten durch

eine Erweiterung der SOP einzuschränken. Es verhindert jedoch keine Angriffe, die keine externe Kommunikation oder das Einfügen von externen Inhalten benötigen. SOMA verhindert auch nicht jene Angriffe, die von einem Partner kommen, der beiderseitig genehmigt wurde.

Zusammenfassend ist zu sagen, dass auf JavaScript basierende Angriffe eine Kommunikation zwischen der kompromittierten Seite und dem durch den Angreifer kontrollierten Webserver benötigen. Das SOMA-Verfahren schränkt diese seitenübergreifende Kommunikation ein. Durch das Verhindern von unangebrachter oder ungenehmigter Kommunikation, können Angriffe wie XSS und XSRF blockiert werden.

Das SOMA-Verfahren schränkt zwar die Kommunikation zu anderen Webservern ein, es beeinflusst jedoch nicht den Zugriff auf persönliche Daten, die der Nutzer dem Anbieter einer Seite übermittelt, wenn er dessen Webanwendung nutzt.

Forscher der Universität von Kalifornien in Davis haben auch die Schwachstellen der SOP im Hinblick auf den Einsatz von Mashups untersucht. Ein Mashup ist eine Webseite, die Inhalte von einer oder mehreren Webseiten kombiniert, beispielsweise das Einbinden von GoogleMaps. Die SOP bietet nur zwei Arten der Vertrauensbeziehung: Einer Seite wird entweder gar nicht vertraut oder ihr wird vollkommen vertraut. Entwickler sind daher gezwungen, sich zwischen Sicherheit und Funktionalität zu entscheiden. Zur Lösung dieses Problems wurde **Object Mashup** (OMash) entwickelt, eine neue Abstraktion und ein Zugriffssteuerungsmodell für die Entwicklung sicherer und trotzdem flexibler Mashupanwendungen [5]. Analog zu Objekten in objektorientierten Programmiersprachen, die nur über ihre öffentlichen Schnittstellen miteinander kommunizieren, werden Webseiten als Objekte behandelt, die eine öffentliche Schnittstelle anbieten. OMash verlässt sich nicht auf die SOP und muss sich so auch nicht mit deren Schwachstellen, wie XSS- und CSRF-Angriffen, auseinandersetzen. Standardmäßig sind alle Inhalte einer Webseite private Daten, auf die nur innerhalb der Seite zugegriffen werden kann. Für die Kommunikation mit anderen Webseiten kann die Webseite eine öffentliche Schnittstelle angeben.

In einem Objekt-Mashup gibt es vier verschiedene Vertrauensverhältnisse, die zwischen dem Integrator und dem Anbieter von Inhalten bestehen können. Dadurch wird definiert, wer auf welche Daten zugreifen darf.

1. **Isoliert:** Kein Zugriff zwischen Integrator und Anbieter
2. **Zugriffsgesteuert:** Begrenzter Zugriff je nach Aufrufer (Überprüfung des Nutzernamens und des Passworts)
3. **Offen:** Vollständiger Zugriff zwischen Anbieter und Integrator, der Integrator und der Anbieter vertrauen sich gegenseitig
4. **Nicht authentifiziert:** Der Anbieter vertraut dem Integrator, aber der Integrator vertraut dem Anbieter nicht

Mit diesem einfachen Modell und der vertrauten Idee der öffentlichen Schnittstellen können Mashupentwickler auf sichere und kontrollierte Art definieren, wie Webseiten von verschiedenen Domains miteinander kommunizieren dürfen. Es

kann feingranular festgelegt werden, wie das Vertrauensverhältnis zwischen Integrator und Anbieter aussieht.

Wie schon bei dem zuvor beschriebenen Verfahren namens SOMA, wird hier der Informationsfluss zwischen verschiedenen Webseiten betrachtet. Die Kommunikation zwischen dem Nutzer und dem Anbieter einer Webanwendung bleibt jedoch außen vor. Der Dienstanbieter kann weiterhin uneingeschränkt auf persönliche Daten zugreifen, die der Nutzer der Webanwendung und somit dem Dienstanbieter übermittelt hat.

SOMA und OMash verringern die Angriffsmöglichkeiten auf die Kommunikation zwischen zwei Webanwendungen. Das trägt dazu bei, dass vertrauliche Inhalte besser vor Dritten geschützt werden können. Obwohl die Verbindung zwischen Nutzer und Dienstanbieter dabei nicht berücksichtigt wird, bietet eine Vereinigung dieser Ansätze mit meinem Verfahren dem Nutzer einen ergänzenden Schutz für seine vertraulichen Daten.

3.2 Verbesserte Funktionsweise

Die beiden in diesem Abschnitt vorgestellten Systeme analysieren die Funktionsweise von Webanwendungen und zeigen auf, wo Fehler in der Funktionalität auftreten können, die zum Beispiel durch unvorhergesehene Wechselwirkungen verschiedener Komponenten verursacht wurden.

Wissenschaftler der Universität von Kalifornien in Santa Barbara haben den **Multi-Modul State Analyzer** (MiMoSA) entwickelt, der Schwachstellen (vulnerabilities) entdeckt, die entstehen wenn mehrere Bausteine einer Webanwendung, auch Module genannt, miteinander interagieren [6]. Es gibt zwei Arten von Schwachstellen, ein Angriff auf den Datenfluss (data-flow) und ein Angriff auf den Arbeitsablauf (workflow). Beispiele für einen Angriff auf den Datenfluss sind SQL-Injection und Cross-Site-Scripting. Bei einem Angriff auf den Arbeitsablauf, verändert der Angreifer den vorgesehenen Programmablauf einer Webanwendung, zum Beispiel um die Nutzerauthentifizierung zu umgehen und erhält somit Zugriff auf Inhalte, die ihm nicht gehören.

MiMoSA kann solche Angriffe verhindern, indem der beabsichtigte Arbeitsablauf (intended workflow) der Webanwendung modelliert wird. Er beinhaltet die Annahmen des Entwicklers über den vorgesehenen Pfad, den der Nutzer beschreitet, während er die Webanwendung ausführt. Zunächst wird ein erweiterter Zustand (extended state) der multimodularen Webanwendung erstellt, er umfasst eine Sammlung von sitzungszugehörigen (session-related) Informationen, auf die durch die verschiedenen Module zugegriffen wird. Danach werden die Interaktionen zwischen der Webanwendung und dem Back-End-System, der Datenbank auf dem Server, analysiert, was dabei hilft, Angriffe auf den Datenfluss zu entdecken. Daraus wird der beabsichtigte Arbeitsablauf der multimodularen Webanwendung abgeleitet, der eine Analysetechnik zur Erkennung von mehrstufigen (multi-step) Angriffen bietet.

MiMoSa schützt multimodulare Webanwendungen vor Angriffen auf ihren beabsichtigten Arbeitsablauf. Dies bewahrt den Nutzer davor, eine Webanwendung

aufzurufen, die sich nicht korrekt verhält. Es hilft ihm aber nicht, die Daten die er der Webanwendung anvertraut, vor missbräuchlichem Zugriff zu schützen.

Forscher der Ohio State Universität in Columbus und des IBM T.J. Watson Forschungszentrums in Hawthorne, New York sowie des IBM CIO Büros haben mit **Splitter** ein Verfahren entwickelt, das Webanwendungen nach ihrer Migration auf neue Plattformen testet [7]. Mit dem Splitter-Verfahren wird überprüft, ob die migrierte Webanwendung fehlerfrei funktioniert, bevor die Umstellung auf diese Plattform stattfindet.

Ein Web-Proxy wird vor die ursprüngliche Webanwendung und die migrierte Webanwendung gesetzt. Der Web-Proxy empfängt die Nutzer-Anfragen (user requests), repliziert sie und sendet sie an die ursprüngliche und an die migrierte Webanwendung. Die Antworten (responses) der beiden Webanwendungen werden von einer Analysemaschine im Web-Proxy miteinander verglichen und dann an den Nutzer weitergeleitet.

Der Ansatz von Splitter erleichtert das Testen von Webanwendungen, die auf eine neue Plattform umgezogen sind. Dadurch können Migrationsprobleme erkannt werden und die Testingenieure können Maßnahmen zur Beseitigung der Probleme durchführen.

Dies führt dazu, dass der Nutzer davor bewahrt wird, eine fehlerhaft migrierte Webanwendung auszuführen. Das ist zwar vorteilhaft für den Nutzer, es hat aber keine Auswirkung auf die Daten, die der Nutzer der Webanwendung anvertraut. Der Anbieter dieser Webanwendung kann auf die Daten des Nutzers in beliebiger Weise zugreifen und der Nutzer hat keinen Einfluss darauf, was mit seinen Daten geschieht. Zu untersuchen wäre in diesem Fall, ob ein Proxy, der zwischen Client und Server installiert wird, auch für Vertraulichkeitsaspekte genutzt werden kann. Man müsste den Proxy so erweitern, dass er zum Server gesendete Daten verschlüsselt und Daten, die zum Client übertragen werden, entschlüsselt. Dafür müsste der Proxy allerdings das Format der Daten kennen und detaillierte Informationen über die Webanwendung besitzen, mit denen er entscheiden kann, welchen Teil der Daten er verschlüsseln kann und welcher Teil unverschlüsselt bleiben muss. Ein solcher Proxy wäre dann allerdings nicht mehr universell für verschiedene Webanwendungen einsetzbar, sondern müsste für jede Webanwendung individuell konfiguriert werden.

Die Lösungsansätze von MiMoSA und Splitter tragen erheblich zur Verbesserung von Webanwendungen bei, weil eine korrekte Funktionsweise die Basis für weitere Schutzmaßnahmen ist. Der korrekte Ablauf einer Webanwendung befasst sich zwar nicht mit Vertraulichkeitsaspekten, er bietet jedoch für den Nutzer eine wertvolle Ergänzung zu der in dieser Arbeit vorgestellten Lösung.

3.3 Schutz vertraulicher Daten

In diesem Abschnitt werden drei Veröffentlichungen vorgestellt, die sich damit auseinandersetzen, wie vertrauliche Daten in Webanwendungen behandelt werden, welchen Angriffen sie ausgesetzt sind und wie sie davor geschützt werden können.

Forscher der Universität von Kalifornien in San Diego haben eine **empirische Studie** durchgeführt, die zeigt wie verbreitet vertraulichkeitsverletzende Informationsflüsse (privacy-violating information flows) in JavaScript-Webanwendungen sind [1]. Untersucht wurde das Auftreten folgender Angriffe: Cookie Diebstahl (cookie stealing), Standort-Entführung (location hijacking), Ausspionieren der Chronik (history sniffing) und Verfolgen des Verhaltens (behavior tracking). Damit so ein Angriff erkannt werden kann, wird das JavaScript der jeweiligen Webseite zur Laufzeit umgeschrieben. Alle Variablen, die vertrauliche Informationen beinhalten, werden mit einer „vertraulich"-Marke (secret) versehen. Sobald eine solche Marke bei der Verwendung einer Variablen auftritt, wird die Aktion blockiert.

Mit diesem Verfahren wurden die 50.000 meistbesuchten Webseiten, laut Alexa Rangliste, auf die genannten Angriffe hin untersucht und es wurde festgestellt, dass auf 485 Webseiten Angriffe, wie das Ausspionieren der Chronik oder das Verfolgen des Verhaltens, stattfinden. Die Firma Alexa ermittelt anhand von Besucherzahlen eine Rangliste der weltweit am häufigsten besuchten Webseiten [2].

Zusammenfassend ist zu sagen, dass in dieser Studie gezeigt wurde, dass die Verletzung der Privatsphäre des Nutzers auf Webseiten ein reales Problem ist und dass man mit diesem Framework, welches das JavaScript einer Webanwendung zur Laufzeit umschreibt, um einen vertraulichkeitsverletzenden Informationsfluss aufzudecken, das Verhalten von Webseiten analysieren kann. Dies ist ein wichtiger Beitrag zur Sicherheit der persönlichen Daten des Nutzers, es hilft ihm aber nicht dabei, diese Daten vor dem Anbieter der ausgeführten Webanwendung, geheim zu halten.

Wissenschaftler des Instituts für Computerwissenschaft und Künstliche Intelligenz am Massachusetts Institut für Technologie haben ein System namens **BFlow** entwickelt, dass mit Hilfe von Informationsfluss-Steuerung (information flow control) die Verwendung von nicht vertrauenswürdigem JavaScript erlaubt, ohne dass die Vertraulichkeit von Daten verletzt wird [8]. Jedes JavaScript wird in einem Browserframe ausgeführt und mehrere Frames werden zu Schutzzonen (protection zones) gruppiert. Jede Schutzzone wird mit einem Etikett (label) versehen. Das Etikett besteht aus einer Menge von Marken (tags), die angeben welche Kategorien von vertraulichen Daten die JavaScript-Dateien in dieser Schutzzone gesehen haben. Wenn Daten versendet werden sollen, überprüft der BFlow Referenzmonitor im Browser, ob das Etikett des Senders eine Teilmenge des Etiketts des Empfängers ist. Ist dies der Fall, findet die Übertragung der Daten statt, sonst wird sie blockiert. Dadurch wird verhindert, dass vertrauliche Daten an einen Empfänger übermittelt werden, der keine Berechtigung hat, diese Daten zu lesen.

Der Nutzer wird mit BFlow davor geschützt, dass seine vertraulichen Daten an ein nicht vertrauenswürdiges JavaScript übermittelt werden. Er muss jedoch weiterhin dem Anbieter der genutzten Webanwendung vertrauen, dass dieser die vertraulichen Daten des Nutzers nicht zweckentfremdet.

Michael Reiher hat in seiner Belegarbeit am Lehrstuhl für Datenschutz und Datensicherheit an der Fakultät Informatik der TU Dresden ein Verfahren entwickelt, dass dem Nutzer erlaubt **Google Mail als sicheres Email-Archiv** zu verwenden [9]. Google scannt den Inhalt der Emails, um dem jeweiligen Nutzer zum Beispiel individuelle Werbung einblenden zu können. Reiher hat eine Lösung entwickelt,

mit der ein Nutzer Google Mail verwenden kann, ohne dass Google Zugriff auf den Inhalt seiner Emails bekommt. In dieser Arbeit wird auf den Emailserver über IMAP zugegriffen. Für das Verbergen des Emailinhalts wird zwischen dem Emailclient und dem Emailserver ein Proxyserver installiert. Dieser verschlüsselt den Header, den Betreff und den Text der Email bevor er diese auf dem Google-Mailserver ablegt, lediglich die E wird von der Verschlüsselung ausgenommen. Umgedreht werden Emails, die vom Server abgerufen werden, durch den Proxyserver entschlüsselt, bevor sie dem Nutzer angezeigt werden. Dadurch wird erreicht, dass die Emails auf dem Google-Mailserver nur verschlüsselt vorliegen und demzufolge nicht gescannt werden können. Sowohl die Verbindung zwischen dem Emailclient und dem Proxyserver, als auch die Verbindung zwischen dem Proxyserver und dem Emailserver wird mittels einer SSL-Verschlüsselung abgesichert.

Mit diesem Verfahren werden die vertraulichen Daten des Nutzers vor dem Dienstanbieter, hier Google Mail, geschützt. Der Nutzer wird also vor einer missbräuchlichen Verwendung seiner Daten bewahrt. Dies ist eine wertvolle Lösung, mit der der Nutzer vertrauliche Daten geheim halten kann. Diese Idee kann grundsätzlich für verschiedene Webanwendungen verwendet werden und ist nicht auf die hier vorliegende Implementierung in einer Emailanwendung begrenzt.

Die empirische Studie macht deutlich, dass Angriffe auf sensible Daten nicht nur theoretische Überlegungen sind, sondern tatsächlich stattfinden. Bis auf die Arbeit von Michael Reiher schützen die vorgestellten Lösungsmöglichkeiten zwar nicht vor einem Angriff des Anbieters der Webanwendung, aber sie können in Kombination mit meinem Verfahren dem Nutzer einen erweiterten Schutz seiner vertraulichen Daten bieten. Michael Reiher hingegen bietet mit seiner Implementierung dem Nutzer eine Möglichkeit, seine Daten vor dem Dienstanbieter geheim zu halten. Dies entspricht dem Ziel, das ich mit meiner Lösung angestrebt habe, es wurde jedoch über einen Proxy realisiert und beschränkt sich auf eine Geheimhaltung von Informationen aus Emails.

3.4 Unzuverlässige Clients

Die in diesem Abschnitt behandelten Verfahren gehen von einer möglichen Kompromittierung der Client-Seite aus und stellen verschiedene Möglichkeiten vor, wie die Auswirkungen der client-seitigen Handlungen beschränkt werden, so dass die Server-Seite vor fehlerhaften oder böswilligen Clients beschützt wird.

Wissenschaftler der Cornell Universität in Ithaca haben eine Methode entwickelt, die Webanwendungen über automatisiertes Partitionieren sicherer macht [10]. Dieses Verfahren wurde **Swift** genannt und verbessert die Vertraulichkeit und Integrität der Informationen in einer Webanwendungen, indem der Anwendungscode in einem Java-ähnlichen Code mit Informationsfluss-Richtlinien (information flow policies) geschrieben wird. Der Compiler nutzt diese Richtlinien für Vertraulichkeit und Integrität, um das Programm automatisch aufzuteilen in JavaScriptcode der auf dem Client ausgeführt wird und Javacode der auf dem Server ausgeführt wird. Mit dieser Aufsplittung wird verhindert, dass ein bösartiger Client auf vertrauliche Informationen auf dem Server zugreifen kann und dass Daten auf dem Server unrechtmäßig durch den Client modifiziert werden.

Mit Swift kann der Nutzer einer Webanwendung davor beschützt werden, dass die Vertraulichkeit und Integrität seiner Daten verletzt wird. Er muss aber weiterhin darauf vertrauen, dass der Anbieter der Webanwendung ordnungsgemäß mit den vom Nutzer übermittelten Daten umgeht und sie nicht zu seinem eigenen Vorteil weiterverwendet. Dennoch kann der Ansatz von Swift ergänzend zu meiner Lösung verwendet werden. Möglicherweise kann diese Instrumentierung auch in der entgegengesetzten Richtung vom Client zum Server eingesetzt werden, um Swift für eine automatische Verschlüsselung zu nutzen.

Forscher der Cornell Universität in New York, der Universität in Neu Delhi und der Microsoft Forschungsabteilung in Redmond haben ein System namens **Ripley** entwickelt, das Webanwendungen automatisch sicherer macht, indem es gleichzeitig eine replizierte Version der Webanwendung ausführt [11]. Bei diesem Verfahren geht man davon aus, dass Berechnungen auf dem Server vertrauenswürdig sind und Berechnungen auf der Client-Seite Opfer eines Angriffs geworden sein können. Damit diesen Client-Berechnungen nicht vertraut werden muss, wird der clientseitige JavaScriptcode repliziert und auf der Server-Seite parallel dazu ausgeführt. Die Ergebnisse werden miteinander verglichen und wenn sich Abweichungen bei den Ergebnissen der Client-Seite zeigen, wird die Verbindung zum Client beendet.
Dieses Verfahren konzentriert sich auf die Integrität verteilter Webanwendungen, nicht auf Vertraulichkeitsaspekte. Die Integrität einer Webanwendung ist für den Nutzer ein wichtiger Faktor, das Ripley-Verfahren hat aber keinen Einfluss auf die Vertraulichkeit der verwendeten Daten gegenüber dem Anbieter der Webanwendung. Der Nutzer ist somit davon abhängig, wie der Anbieter mit den ihm anvertrauten Informationen umgeht.

Wissenschaftler der Universität in Washington haben ein System entwickelt mit dem verschiedene Webanwendungen im Browser voneinander isoliert werden [12]. Monolithische Browser-Architekturen, bei denen alle Seiten von einem einzigen Prozess verarbeitet werden, haben den Nachteil, dass die Verursacher einer Störung oder einer Leistungsminderung nicht erkannt werden und demzufolge auch nicht von den übrigen Webanwendungen isoliert werden können. Der Ausfall einer einzelnen Komponente kann somit zum Absturz des ganzen Browsers führen. Die vorgestellte Multiprozess-Architektur im Google Chrome Browser widmet sich diesen Problemen und verspricht eine Verbesserung der Fehlertoleranz, Robustheit und Leistungsfähigkeit des Browsers. Die Multiprozess-Architektur stellt jeder Instanz einer Webanwendung einen eigenen Rendering-Prozess zur Verfügung, der das HTML-Rendering und die JavaScript-Verarbeitung durchführt. Zudem gibt es einen Prozess namens Browser-Kernel, der sich um die Speicherverwaltung, die Netzwerkverbindung und die Nutzerschnittstelle kümmert. Ein weiterer Prozess führt die Browser-Plugins aus. Damit verbessert sich die Fehlertoleranz des Browsers, denn eine Störung in einer HTML-Rendering-Engine verursacht lediglich den Ausfall des zugehörigen Prozesses, die Instanzen der anderen Webanwendungen bleiben davon unberührt. Genauso können Instanzen von Webanwendungen identifiziert und gestoppt werden, die Ressourcen wie Arbeitsspeicher, CPU oder die Netzwerkverbindung übermäßig beanspruchen. Durch die parallele Ausführung der verschiedenen Prozesse verbessert sich die Reaktionsfähigkeit der Webanwendungen und der Nutzer nimmt weniger Verzögerungen wahr.

Diese Isolierung von Webanwendungen durch die Verwendung mehrerer Prozesse erhöht die Fehlertoleranz und Leistungsfähigkeit des Browser für den Nutzer. Die isolierten JavaScript-Engines bilden die Grundlage für eine separate Verschlüsselung der privaten Nutzerdaten in verschiedenen Webanwendungen. In meiner prototypischen Implementierung wurde der Chrome Browser mit diesen isolierten JavaScript-Engines verwendet, um dem Nutzer eine separate Verschlüsselung pro Webanwendung zur Verfügung zu stellen.

Auch wenn sich diese Verfahren nicht mit der Vertraulichkeit von sensiblen Nutzerdaten beschäftigen, so sind sie doch eine wertvolle Ergänzung zu meinem Ansatz, weil sie dem Nutzer ein zusätzliches Maß an Sicherheit geben, wenn der Server, auf dem seine vertraulichen Daten liegen, vor Angriffen von unzuverlässigen Clients geschützt ist.

3.5 Speicherung auf unzuverlässigen Servern

In diesem Abschnitt werden drei Verfahren vorgestellt, die sich mit der Speicherung von Daten auf nicht vertrauenswürdigen Servern auseinandersetzen und Lösungen anbieten, die trotz dieser Gegebenheiten dem Nutzer die Integrität seiner Daten gewährleisten.

Wissenschaftler der Universität New York haben ein Netzwerkdateisystem namens **Secure Untrusted Data Repository** (SUNDR) entwickelt, das Nutzern die Möglichkeit gibt, ihre Daten sicher auf einem nicht vertrauenswürdigen Server zu speichern [13]. Mit SUNDR kann der Nutzer erkennen, ob von einem anderen Nutzer oder dem Serverbetreiber ein Versuch unternommen wurde, seine Dateien zu modifizieren. Die Anwendungen auf der Client-Seite greifen auf das Dateisystem zu, indem sie entweder eine Abruf- (fetch) oder Modifizierungs-Operation (modify) ausführen. Mit der Abruf-Operation wird der Inhalt einer Datei aufgerufen und mit der Modifizierungs-Operation wird den anderen Nutzern ein neuer Systemstatus bekannt gegeben. Auf der Server-Seite gibt es einen Konsistenzserver, der alle Operationen protokolliert und einen Blockserver, auf dem die Dateien, indexiert über einen Hashwert, gespeichert werden. Der Client versieht jede Operation mit einer Signatur, die neben der aktuellen Operation auch das Protokoll aller bisherigen Operationen umfasst. Wenn der Server sich korrekt verhält, spiegelt der Inhalt einer Datei nach einer Abruf-Operation genau die davor stattgefundenen Operationen wieder. Diese Eigenschaft wird Abruf-Modifizierungs-Konsistenz (fetch-modify consistency) genannt.

Wenn ein bösartiger Nutzer oder Server Dateien modifiziert, führt das zu inkonsistenten Protokollen. Dadurch merkt der Nutzer, dass eine nicht autorisierte Modifizierung stattgefunden hat. Auch wenn der Server die Modifizierungs-Operation des einen Nutzers vor der Abruf-Operation eines anderen Nutzers verbergen will, so gelingt ihm dies nur, wenn er den beiden Nutzern ein unterschiedliches Protokoll der erfolgten Operationen vorlegt. Sonst würden die Nutzer den Angriff bei der darauffolgenden Operation entdecken. Das wird als gespaltene Konsistenz (fork consistency) bezeichnet. Nutzer die auf einem Seitenkanal miteinander über die modifizierten Dateien kommunizieren, können solch eine gespaltene Konsistenz leicht feststellen.

Mit dem Netzwerkdateisystem SUNDR können Nutzer sich darauf verlassen, dass selbst ein nicht vertrauenswürdiger Server ihnen immer einen korrekten Zustand des Dateisystems anzeigt. Der Server verhält sich entweder korrekt, oder die Nutzer können, indem sie über einen Seitenkanal miteinander kommunizieren feststellen, dass ihnen unterschiedliche Zustände des Dateisystems vorgelegt wurden.

Mit diesem Verfahren wird die Integrität der Daten geschützt, was ein wichtiger Vorteil für den Nutzer ist, nicht jedoch die Vertraulichkeit. Der Inhalt der Dateien steht dem Server ohne Einschränkung zur Verfügung und der Nutzer kann nicht beeinflussen, für welche Zwecke seine persönlichen Daten verwendet werden.

Ein Verfahren namens **Provable Data Possession** (PDP) wurde von Forschern der Universitäten in Baltimore und Berkeley entwickelt [14]. Es ermöglicht Nutzern, die ihre Daten auf nicht vertrauenswürdigen Servern gespeichert haben, zu prüfen, ob der Server die Originaldaten besitzt oder einen Teil der Daten gelöscht hat. Der Nutzer muss dafür seine Daten nicht abrufen. Das Verfahren erzeugt einen probabilistischen Besitzbeweis (proof of possession) durch eine Menge zufälliger Stichproben von Blöcken auf dem Server. Der Client behält eine konstante Menge von Metadaten, um diesen Beweis überprüfen zu können. Das PDP-Verfahren besteht aus vier Algorithmen:

Schlüsselerzeugung (key generation) - Ein probabilistischer Algorithmus zur Schlüsselerzeugung, der vom Client ausgeführt wird und durch die Eingabe eines Sicherheitsparameters ein zusammengehöriges Schlüsselpaar erzeugt, bestehend aus einem öffentlichen (public) und einem privaten Schlüssel (private key).

Blockmarke (tag block) - Vom Client ausgeführter Algorithmus, der für einen Block Überprüfungsmetadaten erzeugt. Die Eingabewerte sind der öffentliche und der private Schlüssel und der Dateiblock. Der Ausgabewert sind die Überprüfungsmetadaten, wobei für jeden Block eine Blockmarke erzeugt wird.

Beweisgenerierung (proof generation) - Der Server führt diesen Algorithmus aus, um den Besitzbeweis (proof of possession) zu erbringen. Mit dem öffentlichen Schlüssel, einer geordneten Menge von Dateiblöcken, einer Herausforderung (challenge) und einer geordneten Sammlung der Blockmarken wird der Besitzbeweis generiert.

Beweisprüfung (check proof) - Ein Algorithmus der vom Client ausgeführt wird, um zu überprüfen, ob der Besitzbeweis gültig ist oder nicht. Als Eingabewerte werden der öffentliche und der private Schlüssel, die Herausforderung und der Besitzbeweis verwendet. Die Ausgabe besteht aus der Bestätigung oder Ablehnung des Beweises, für die in der Herausforderung angegebenen Blöcke.

Das PDP-Verfahren besteht aus zwei Phasen, der Einrichtung (setup) und der Herausforderung (challenge). In der Einrichtungsphase besitzt der Client die Datei, ruft die Schlüsselgenerierung und die Erzeugung der Blockmarken auf. Er speichert das Schlüsselpaar und sendet den öffentlichen Schlüssel, die Datei sowie die Blockmarken an den Server zur Speicherung. Danach löscht er die Datei und die

Blockmarken in seinem lokalen Speicher. In der zweiten Phase erstellt der Client eine Herausforderung. Er legt fest, für welche Teilmenge von Blöcken er einen Besitzbeweis verlangt und sendet diese Herausforderung an den Server. Mit der Herausforderung generiert der Server einen Besitzbeweis, den er zurück zum Client schickt. Der Client überprüft daraufhin die Gültigkeit des Besitzbeweises und erkennt je nach Ergebnis, ob der Server die Originaldaten besitzt oder einen Teil von ihnen gelöscht hat.

Dieses Verfahren gibt Clients die Möglichkeit zu überprüfen, ob ihre Daten, die sie auf einem nicht vertrauenswürdigen Server gespeichert haben, noch im originalen Zustand vorhanden sind oder teilweise gelöscht wurden. Der Beweis wird über einen stichprobenartigen Zugriff auf die Blöcke der Dateien erstellt. Der Client muss nicht auf die gesamten Daten zugreifen und kann so in sehr geringer Zeit erfahren, ob seine Daten auf dem entfernten Server noch vollständig sind.

Dieser Beweis über die Vollständigkeit seiner Daten ist für den Nutzer sehr hilfreich, er bewahrt ihn aber nicht vor einer unsachgemäßen Verwendung seiner Daten. Das PDP-Verfahren schützt nicht die Vertraulichkeit der Daten.

Wissenschaftler der Universität von Texas in Austin haben ein Cloud-Speichersystem namens **Depot** entwickelt, das die Annahmen über die Zuverlässigkeit der beteiligten Server und Clients minimiert [15]. Depot kann fehlerhaftes oder böswilliges Verhalten einer beliebigen Anzahl von Servern oder Clients tolerieren und korrekten Clients trotzdem Sicherheits- und Lebendigkeitsgarantien anbieten. Diese Garantien zur Integrität und Verfügbarkeit der Daten werden bereitgestellt, indem eine zweischichtige Architektur verwendet wird. Zuerst wird sichergestellt, dass die Updates, die von den korrekten Clients wahrgenommen werden durchweg geordnet sind. Danach realisiert Depot basierend auf dieser konsistenten Ordnung der Updates, Protokolle, die weitere wünschenswerte Eigenschaften liefern, wie Konsistenz, Dauerhaftigkeit, Wiederherstellbarkeit oder die Erkennung von veralteten Daten.

Der Nutzer erhält mit Depot Integritäts- und Verfügbarkeitsgarantien für seine Daten, nicht jedoch eine vertrauliche Behandlung derselben.

Die in diesem Abschnitt vorgestellten Lösungen sind orthogonal zu meiner Arbeit, sie können jedoch in Kombination mit der von mir vorgestellten Lösung, den Nutzer dabei unterstützen, seine Daten sicherer in einer Webanwendung zu verwenden.

4 Entwurf

Webanwendungen bestehen aus einem client-seitigen Teil, der auf dem Computer des Nutzers ausgeführt wird und einem server-seitigen Teil, der durch die Webseite des Dienstanbieters zur Verfügung gestellt wird. Während die Webanwendungen ausgeführt werden, wie beispielsweise eine ToDo-Liste (Aufgabenliste) und ein Kalender (siehe Abbildung 4.1), senden und empfangen der Server und der Client die individuellen Daten des Nutzers. Der Nutzer möchte seine Webanwendungen von verschiedenen Computern aus nutzen können, zu Hause, an seinem Arbeitsplatz oder in einem Internetcafé. Für eine ortsunabhängige Verfügbarkeit müssen die Daten der jeweiligen Webanwendung auf der Server-Seite beim Dienstanbieter gespeichert sein. Hier werden die individuellen Daten verschiedener Nutzer in einer Datenbank abgelegt. Jeder Nutzer ruft mit seinem Browser auf der Client-Seite die Webseite des Dienstanbieters auf.

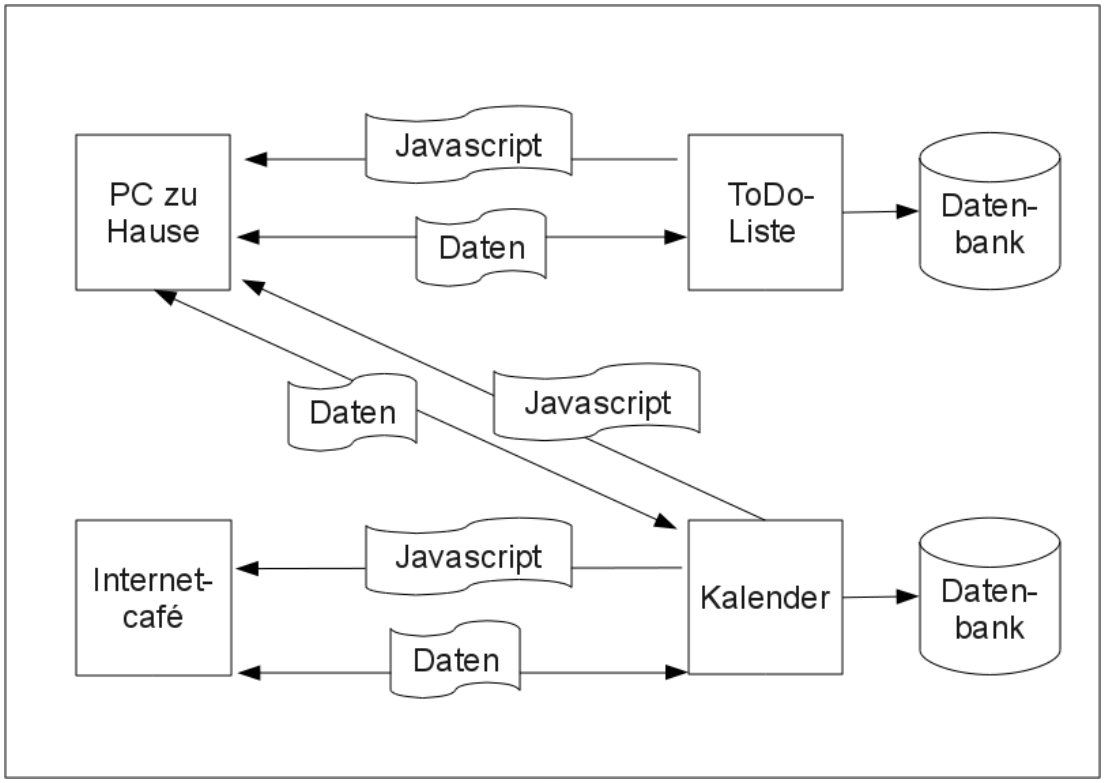

Abbildung 4.1: Datenzugriff

Der client-seitige Teil der Webanwendung wird als JavaScript in den Browser geladen und ausgeführt. Dadurch werden die individuellen Daten des angemeldeten Nutzers auf die Client-Seite übertragen. Während der Nutzer mit seiner Webanwendung arbeitet, kann er Daten erzeugen, verändern und löschen. Dies führt da-

zu, dass auch Daten von der Client-Seite zur Server-Seite übertragen werden und die Datenbank somit aktualisiert wird. Dadurch ist der Nutzer in der Lage, jederzeit über das Internet auf seinen aktuellen Datenbestand zuzugreifen, indem er die Webseite des Dienstanbieters aufruft. Mit dem Loginvorgang identifiziert sich der Nutzer gegenüber dem Dienstanbieter, so dass dieser weiß, welche individuellen Daten an den Browser zu übertragen sind. Dadurch kann auf der Server-Seite gewährleistet werden, dass der angemeldete Nutzer Zugriff auf seine individuellen Daten erhält, nicht jedoch auf die Daten anderer Nutzer. So wie es auf der Server-Seite verschiedene Nutzer mit individuellen Daten gibt, muss der Browser auf der Client-Seite zwischen mehreren Webanwendungen unterscheiden, um die darin enthaltenen Daten voneinander zu isolieren.

Im Abschnitt 2.2 wurde bereits beschrieben, wie ein Angreifer unberechtigten Zugriff auf private Nutzerdaten erhalten kann. Im Folgenden wird nun erläutert, mit welchen Maßnahmen sich der Nutzer vor den Angriffen des Dienstanbieters schützen kann. Im darauffolgenden Abschnitt wird analysiert, welche Webanwendungen für einen verbesserten Datenschutz geeignet sind. Im Weiteren wird dargestellt, welche Anpassungen in der Webanwendung und im Browser notwendig sind. Das Entwurfskapitel endet mit einem Abschnitt der zeigt, wie eine Webanwendung mit verbessertem Datenschutz abläuft.

4.1 Verbesserter Datenschutz

Grundsätzlich kann man Daten mit einer Verschlüsselung vor einem unberechtigten Zugriff schützen. Wie bereits in Abschnitt 2.2 beschrieben, können Angriffe im Namen des Nutzers oder auf den Datenverkehr durch den Einsatz einer verschlüsselten Verbindung verhindert werden.

Auch vor dem Dienstanbieter kann der Nutzer seine Daten mit Hilfe einer Verschlüsselung schützen. Weil aber der Dienstanbieter die Webanwendung liefert, in der die Daten verschlüsselt werden, kann das Vertrauen in den Dienstanbieter nur bis zu einem gewissen Grad reduziert werden.

Ein Dienstanbieter der sich mit einer neuen Webanwendung von den anderen Dienstanbietern abgrenzen will, kann dem Nutzer mehr Privatsphäre anbieten, als dieser normalerweise bei derartigen Webanwendungen vorfindet. Der Dienstanbieter muss dafür den Bereich der Privatsphäre vergrößern, indem er sich selbst aus dem Kreis der möglichen Angreifer herausnimmt. Dies kann er erreichen, indem er darauf verzichtet, die Daten die der Nutzer ihm anvertraut im Klartext lesen zu können. So stellt er dem Nutzer eine Webanwendung zur Verfügung, die seine Privatsphäre respektiert.

Der Nutzer muss dann nur seinem Browser vertrauen und schließt mit dem Dienstanbieter einen Vertrag darüber, wie eine Webanwendung mit verbessertem Datenschutz aussieht. Der Nutzer und der Dienstanbieter einigen sich darauf, dass die individuellen Daten des Nutzers nur verschlüsselt auf dem Server des Dienstanbieter abgelegt werden und es wird sichergestellt, dass der Dienstanbieter den Schlüssel nicht kennt. Um diese Garantien technisch umzusetzen, findet die Ver- und Entschlüsselung der Daten nur im Browser statt. Dafür braucht der Nutzer in seinem Browser eine Verschlüsselungskomponente, die eine Verschlüsselungsfunktion und eine Entschlüsselungsfunktion bereit stellt. Die Verpflichtung

des Dienstanbieters besteht darin, dass er diese beiden Funktionen in den client-seitigen Anteil der Webanwendung integriert.

4.2 Eignung von Webanwendungen

Viele Webanwendungen bieten ihre Funktionalität über server-seitiges PHP und nicht über client-seitiges JavaScript an. Eine client-seitige Absicherung der Daten im Browser ist in diesem Fall nicht anwendbar.

In gleicher Weise ungeeignet sind Desktopanwendungen, die die Daten des Nutzers lokal auf seinem Rechner speichern. Eine ortsunabhängige Verfügbarkeit der Daten, wie in Abbildung 4.1 dargestellt, kann mit diesen Anwendungen nicht realisiert werden.

Für einen verbesserten Datenschutz kommen nur Webanwendungen in Frage, deren Funktionalität über ein client-seitiges JavaScript realisiert wird und bei denen die in der Webanwendung verwendeten Daten auf der Server-Seite gespeichert werden. Webanwendungen wie zum Beispiel Facebook, bei denen die server-seitig gespeicherten Daten anderen Nutzern angezeigt werden sollen, würden durch eine Verschlüsselung der Daten ihre Funktionalität verlieren und sind somit ebenfalls nicht für einen verbesserten Datenschutz geeignet.

Wenn die Daten auf der Server-Seite nicht nur gespeichert, sondern auch verarbeitet werden, um einen Teil der Funktionalität der Webanwendung zu realisieren, kommt es darauf an, ob der Server auch mit den verschlüsselten Daten arbeiten kann. Falls das nicht der Fall ist, kommen sie für eine Verbesserung des Datenschutzes durch Verschlüsselung nicht in Frage.

Es gibt zahlreiche frei verfügbare JavaScript-Frameworks mit denen man sich eine Webanwendung nach eigenen Vorstellungen zusammenstellen kann. Bei den meisten liegt der Schwerpunkt jedoch auf einer umfangreichen graphischen Gestaltung und nicht auf einem Datenaustausch zwischen dem Browser und dem Webserver. Deshalb sind auch sie für einen verbesserten Datenschutz nicht geeignet.

Das Open-Source-Framework Sproutcore [16] hat sich schließlich als geeignet herausgestellt, weil die Funktionalität der Webanwendung mit einem client-seitigen JavaScript zur Verfügung gestellt wird und die Daten server-seitig gespeichert werden. Als Demonstrationsbeispiel bietet Sproutcore eine ToDo-Liste (Aufgabenliste) [17] an, die als Beispielanwendung für die prototypische Implementierung ausgewählt wurde.

4.3 Instrumentierung der Webanwendung

Die Instrumentierungen in der Webanwendung beziehen sich auf die Vertraulichkeit und die Integrität der Daten des Nutzers.

4.3.1 Vertraulichkeit

In der Webanwendung müssen Sendeoperationen, die Nutzerdaten vom Client zum Server übertragen, mit einer zusätzlichen Verschlüsselungsfunktion versehen werden. Passend dazu werden Empfangsoperationen, mit denen Nutzerdaten vom

Server heruntergeladen werden, um eine Entschlüsselungsfunktion ergänzt. Sobald alle HTTP-Anfragen identifiziert sind, untersucht man, in welcher Struktur die Daten vorliegen. Es gilt zu unterscheiden, was reine Metadaten sind und woraus die konkreten Nutzerdaten bestehen.

Metadaten werden von der Client- und der Server-Seite zur Verwaltung benötigt, um zum Beispiel einen einzelnen Datensatz zu identifizieren. Diese Menge ist disjunkt zur Menge der Nutzerdaten und besteht aus Elementen, die nicht verschlüsselt werden können ohne die Funktionalität der Webanwendung zu beeinträchtigen.

Nutzerdaten bestehen aus individuellen Inhalten, die der Nutzer in Textform erstellt hat und die ohne Weiteres verschlüsselt werden können.

Strukturgebundene Informationen sind eine besondere Art von Nutzerdaten, die Inhalte beschreiben, wie eine konkrete Uhrzeit oder einen booleschen Wert. Sie eignen sich aufgrund ihres Wertebereichs nur begrenzt für eine Verschlüsselung. Hier muss eine individuelle Strategie zur Verschleierung des Informationsgehalts angewendet werden.

Der Dienstanbieter muss in seiner Webanwendung die soeben genannten Daten identifizieren und sie im JavaScript mit den entsprechenden Verschlüsselungs- und Verschleierungsverfahren versehen. Unterschieden werden muss an dieser Stelle, ob sich die Daten auf dem Weg zum Server oder zum Client befinden. Dementsprechend werden sie einer Ver- oder Entschlüsselung unterzogen.

Bei der ToDo-Liste von Sproutcore beispielsweise besteht die zu übertragende Datenstruktur aus dem Datenbankindex des Eintrages, einem Textfeld für die Beschreibung der Aufgabe und einem booleschen Wert für den Status. Anhand dieser drei Einzeldaten wird ersichtlich, in welcher Form eine Verschlüsselung möglich ist und wodurch sie begrenzt ist. Der Datenbankindex ist ein nicht zu verschlüsselndes Metadatum. Die Beschreibung der Aufgabe ist vollständig verschlüsselbar und der Status des Eintrages muss individuell verborgen werden.

Auf der Client-Seite muss, entsprechend zu den Instrumentierungen des Dienstanbieters, eine generische Anpassung der JavaScript-Engine vorgenommen werden, so dass die erforderlichen Verschlüsselungsvorgänge für die Nutzerdaten und die Verschleierungsmaßnahmen für die strukturgebundenen Informationen durchgeführt werden können.

Passend dazu wird die JavaScript-Engine um Entschlüsselungsfunktionen erweitert, die die Nutzerdaten und die strukturgebundenen Informationen wieder in ihren ursprünglichen Zustand zurück versetzen.

4.3.2 Integrität

Mit der Verschlüsselung der persönlichen Daten des Nutzers wird dem Dienstanbieter die Möglichkeit genommen, auf den Inhalt der Daten zuzugreifen. Eine Verschlüsselung hindert den Dienstanbieter jedoch nicht daran, die Integrität der Daten zu verletzen. Er hat immer noch die Möglichkeit, einzelne Einträge des Nutzer zu löschen, Kopien von Einträgen hinzuzufügen, oder einen aktuellen Eintrag durch einen veralteten zu ersetzen.

Wenn man die Daten auch in dieser Hinsicht vor dem Dienstanbieter schützen will, müsste man grob skizziert folgendermaßen vorgehen. Die einzelnen Einträge in der Webanwendung werden mit einer Prüfsumme versehen und für den Gesamtzustand der Daten in der Webanwendung wird eine Prüfsumme ermittelt, mit der überprüft werden kann, ob Einträge gelöscht, hinzugefügt oder verändert wurden. Dafür wird bei jeder Änderung eines Eintrages, die zugehörige Prüfsumme aktualisiert und die Prüfsumme des Gesamtzustandes erstellt. Diese kann man zum Vergleich auf dem USB-Stick des Nutzer abspeichern, oder in Form von Signaturen den Daten hinzufügen. So kann der Browser beim Start der Webanwendung, wenn er die Daten von der Server-Seite herunterlädt, überprüfen, ob die Einträge unverändert geblieben sind oder vom Dienstanbieter manipuliert wurden.

Der in dieser Arbeit vorgestellte verbesserte Datenschutz konzentriert sich auf den Schutz der Vertraulichkeit der Daten und nicht auf den Schutz der Integrität der Daten. Zudem können Aussagen über die Integrität des Gesamtzustandes der Daten nicht immer generisch getroffen werden. Zum Beispiel werden bei einer Webanwendung, die einen Kalender realisiert, am Anfang eventuell nicht alle Daten auf einmal in den Browser geladen, sondern nur die Termine der aktuellen und der nächsten Woche. Schutzmaßnahmen für die Integrität aller Daten können also nicht immer generisch realisiert werden und müssen somit anwendungsspezifisch implementiert werden.

4.4 Anpassungen im Browser

Die im Abschnitt 4.3.1 geschilderte Instrumentierung der Webanwendung kann nur realisiert werden, wenn im Browser auf der Client-Seite die Funktionalität zur Ver- und Entschlüsselung zur Verfügung gestellt wird. Zudem muss der Browser seinen Nutzer dabei unterstützen, erkennen zu können, ob ein verbesserter Datenschutz vom Dienstanbieter korrekt umgesetzt wurde.

4.4.1 Verschlüsselung

Die JavaScript-Engine des Browsers wird generisch um eine Verschlüsselungsfunktion (enCrypt) und eine Entschlüsselungsfunktion (deCrypt) erweitert. Diese beiden Funktionen nehmen den jeweils zu transformierenden Text als Argument und geben je nach Funktion den verschlüsselten oder den entschlüsselten Text zurück. Abbildung 4.2 zeigt die Ver- und Entschlüsselung des Wortes „Geheimnis“.

Abbildung 4.2: Verwendung der Ver- und Entschlüsselung

Wie bereits in der Abbildung 4.1 gezeigt wurde, soll der Nutzer verschiedene Webanwendungen mit verbessertem Datenschutz parallel ausführen können. Für eine unabhängige Verschlüsselung der jeweiligen Daten erhält jede Webanwendung einen eigenen Schlüssel, der in einer Schlüsselbibliothek hinterlegt wird. Die Schnittstelle zu dieser Schlüsselbibliothek besteht aus den zwei Funktionen keyPut und keyGet und wird dem Browser zur Verfügung gestellt. Mit diesen beiden Funktionen kann die JavaScript-Engine neu generierte Schlüssel abspeichern oder den passenden Schlüssel der gerade ausgeführten Webanwendung auswählen.

4.4.2 Authentifizierung

Neben der Verschlüsselungskomponente umfasst die Erweiterung des Browsers auch eine Überprüfung der Webanwendung mit einer graphischen Rückmeldung. Dadurch wird der Nutzer davor bewahrt, eine manipulierte Webanwendung auszuführen. Um zu überprüfen, ob sich der Dienstanbieter an die vertragliche Vereinbarung hält, gibt es verschiedene Möglichkeiten.

Der auf den ersten Blick einfachste Weg ist eine Einigung zwischen dem Nutzer und dem Dienstanbieter auf eine gültige Version des Quellcodes. Der Browser überprüft nach dem Download, ob genau das erwartete JavaScript gesendet wurde und lehnt es gegebenenfalls ab. Der Dienstanbieter wird dadurch aber erheblich eingeschränkt, weil er sich bei jeder technischen Änderung mit dem Nutzer erneut auf eine gültige Version einigen muss. Dabei muss immer wieder die Entscheidung getroffen werden, ob sich das JavaScript korrekt verhält oder nicht.

Eine Überprüfung und Verifizierung, ob bestimmte Sicherheitsrichtlinien im Quellcode einer Anwendung eingehalten werden, bietet das „Proof-Carrying-Code“-Verfahren [18]. Das JavaScript selbst würde damit einen Beweis enthalten, dass zum Beispiel alle unverschlüsselten Sendeversuche verhindert werden. Ähnlich wie im Abschnitt 4.3.1 beschrieben, bietet dieses Verfahren eine Möglichkeit, Anwendungen auf ihren Umgang mit sensiblen Daten hin zu überprüfen und gegebenenfalls zu verbessern. Die Realisierung dieses Verfahrens für JavaScriptcode wäre jedoch so umfangreich, dass es den Rahmen dieser Arbeit sprengen würde.

Letztendlich interessiert den Nutzer nur das Ergebnis, also bietet es sich an, die Überprüfung des Quellcodes einem Dritten zu überlassen, der dem Dienstanbieter ein korrektes JavaScript bescheinigen kann. Der Dienstanbieter instrumentiert die Webanwendung so, dass vor jedem Senden der Daten an ihn eine Verschlüsselung stattfindet. Damit der Nutzer sich nicht auf ein Versprechen zur erhöhten Privatsphäre vom Dienstanbieter verlassen muss, wird die Webanwendung von einer Zertifizierungsstelle auf diese Eigenschaft hin untersucht und je nach Ergebnis mit einem Zertifikat versehen oder nicht. Somit kann der Nutzer vor der Ausführung der Webanwendung die Gültigkeit des Zertifikats bei der Zertifizierungsstelle abfragen. Wenn das Zertifikat ungültig ist, wird die Webanwendung gestoppt.

Der Nutzer vertraut also darauf, dass die Zertifizierungsstelle überprüft hat, ob der Dienstanbieter in der Webanwendung die im Abschnitt 4.3.1 beschriebenen Instrumentierungen korrekt und vollständig durchgeführt hat. Zudem verlässt sich der Nutzer auf seinen Browser, dass dieser die in der Webanwendung verwendeten Daten korrekt ver- und entschlüsselt und dabei auch die Daten aus verschiedenen

Webanwendungen voneinander isoliert. Der Nutzer kann sich somit darauf verlassen, dass der Dienstanbieter seine Daten nicht mehr im Klartext lesen kann.

Im folgenden Abschnitt wird beschrieben, wie eine Webanwendung mit verbessertem Datenschutz abläuft.

4.5 Ablauf einer instrumentierten Webanwendung

Dieser Abschnitt erklärt zunächst den Registriervorgang, den der Nutzer durchlaufen muss, bevor er eine Webanwendung mit verbessertem Datenschutz zum ersten Mal ausführen kann. Danach wird beschrieben, wie die Ausführung einer solchen Webanwendung abläuft.

4.5.1 Registrierung

Ein neuer Nutzer, der eine Webanwendung mit verbessertem Datenschutz ausführen möchte, ruft zunächst die Webseite dieser Webanwendung auf und registriert sich dort.

Sobald ein registrierter Nutzer die Webanwendung aufruft und sich erfolgreich anmeldet hat, erhält er den zertifizierten JavaScriptcode. Daraufhin kann er bei der Zertifizierungsstelle abfragen, ob das Zertifikat gültig ist und falls dies der Fall ist, die Webanwendung ausführen.

Bei der erstmaligen Verwendung generiert der Browser auf der Client-Seite einen Schlüssel, mit dem er ab sofort die Daten des Nutzers ver- und entschlüsselt. Dieser Schlüssel wird vom Browser auf dem USB-Stick des Nutzers abgelegt, so dass nur der Nutzer in Besitz des USB-Sticks die Webanwendung ausführen kann. Weil die Schlüssel für die verschiedenen Webanwendungen mit verbessertem Datenschutz auf dem USB-Stick des Nutzers gespeichert werden, ist dieser in der Lage, von jedem beliebigen Rechner aus seine Webanwendungen aufzurufen. Er benötigt dazu lediglich einen Browser, der die im Abschnitt 4.4 beschriebene Erweiterung realisiert und seinen USB-Stick.

4.5.2 Ausführung

Abbildung 4.3 zeigt den Ablauf einer Webanwendung mit verbessertem Datenschutz (VD). Dabei wird an mehreren Stellen überprüft, ob der richtige Nutzer die korrekte Webanwendung ausführt. Als Erstes startet der registrierte Nutzer seinen erweiterten Browser und teilt ihm mit, wo sich seine Schlüssel befinden. Findet der Browser beim Start der Webanwendung (1) keinen Schlüssel (Key), prüft er, ob das JavaScript (JS) der auszuführenden Webanwendung ein Zertifikat enthält (2). Dies zeigt ihm, ob es sich um eine Webanwendung mit oder ohne verbesserten Datenschutz handelt. Eine Webanwendung ohne verbesserten Datenschutz wird vom Browser ohne weitere Prüfungen ausgeführt (3). Fehlt jedoch der Schlüssel und es liegt eine Webanwendung mit verbessertem Datenschutz vor, erhält der Nutzer eine graphische Rückmeldung, dass er versucht hat, eine Webanwendung auszuführen für die er keine Berechtigung hat (4).

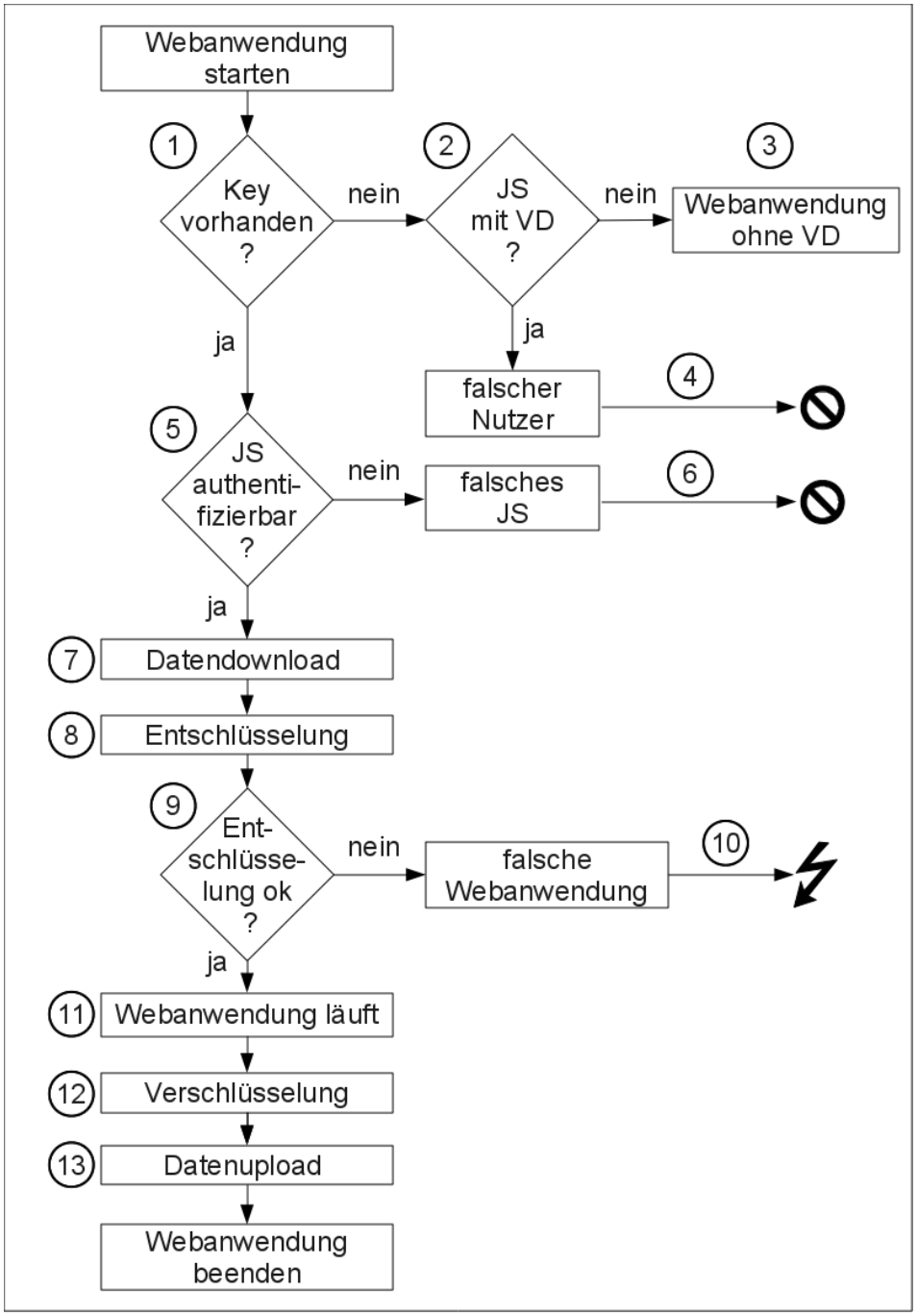

Abbildung 4.3: Bedingte JavaScript-Ausführung

Findet der Browser den passenden Schlüssel, überprüft er als Nächstes, ob das dem JavaScriptcode angefügte Zertifikat gültig ist (5). Wenn das nicht der Fall ist, erhält der Nutzer eine Warnmeldung, die besagt, dass es sich um keine gültige Version seiner Webanwendung handelt und er wird daran gehindert, die Webanwendung weiter auszuführen (6).

Bei einer erfolgreichen Überprüfung des Zertifikats werden die persönlichen Daten des Nutzers in den Browser geladen (7) und entschlüsselt (8). Es wird überprüft,

ob die Entschlüsselung erfolgreich durchgeführt wurde (9). Wenn eine andere Webanwendung mit einem anderen Schlüssel versucht, diese Daten zu entschlüsseln, schlägt dieser Versuch fehl und es werden falsch entschlüsselte und damit unlesbare Daten erzeugt (10). Werden die Daten erfolgreich entschlüsselt, kann der Nutzer mit der Arbeit in seiner Webanwendung beginnen (11). Alle Daten die hinzugefügt oder angepasst werden, durchlaufen die Verschlüsselung (12) und werden zum Server übertragen (13). Dazu wird bei jeder Operation die Identität der Webanwendung ermittelt, die gerade Daten verschlüsseln will, so dass immer der dazugehörige Schlüssel ausgewählt wird und eine korrekte Ver- und Entschlüsselung stattfindet.

5 Implementierung

Die Implementierung besteht aus einer Verschlüsselungserweiterung des Browsers und aus einer Verschlüsselungsinstrumentierung der Webanwendung (siehe Abbildung 5.1). Dem Browser auf der Client-Seite wird dafür ein Key-Value-Store zur Verfügung gestellt (1) und die JavaScript-Engine (2) wird um die zwei Kryptographiefunktionen `enCrypt` und `deCrypt` erweitert. Das auf der Server-Seite gespeicherte JavaScript (JS) der Webanwendung wird mit diesen beiden Funktionen instrumentiert (3) und mit einem Zertifikat (4) für die Authentifizierung versehen. Im Folgenden werde ich diese vier Teilbereiche näher erläutern.

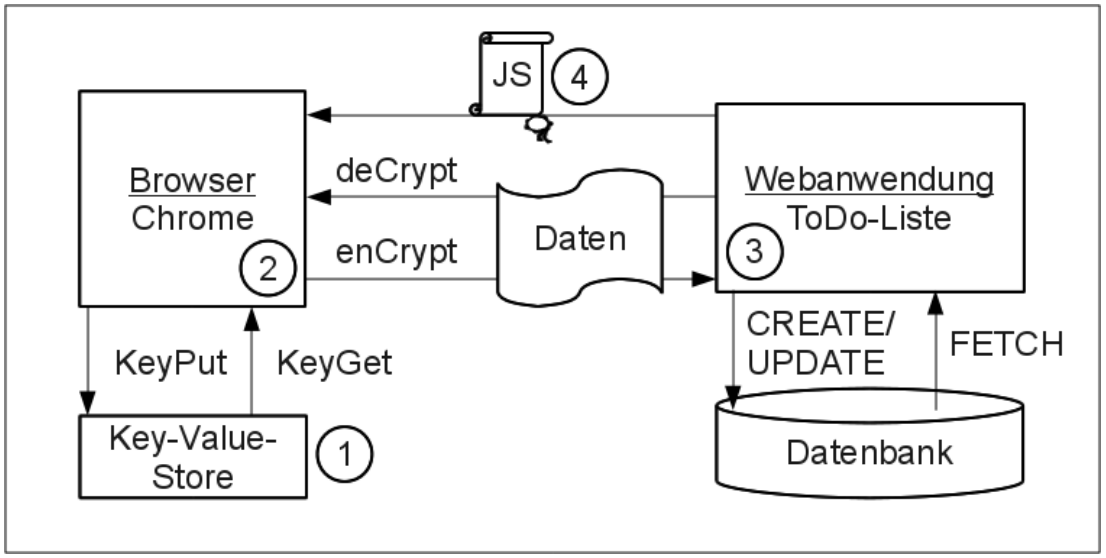

Abbildung 5.1: Implementierungskomponenten

5.1 Persistenter Key-Value-Store

Der Key-Value-Store (KVStore) repräsentiert einen Schlüsselbund, den der Nutzer dem Browser zur Verfügung stellt, so dass passend zur jeweiligen Webanwendung ein Schlüssel ausgewählt wird, mit dem die Kryptographiekomponente die Daten ver- und entschlüsselt.

Die Schnittstelle des KVStores besteht aus einer Funktion zum Speichern eines Schlüssels und einer Funktion zum Auslesen eines bestimmten Schlüssels. Zur Laufzeit werden die Schlüssel in einer Hashtabelle abgelegt. Im Hintergrund wird gleichzeitig für jeden Schlüssel eine Datei angelegt, so dass der Wert persistent vorhanden ist.

KeyPut(Key,Value) Es wird eine Datei mit dem Namen der Webanwendung als `Key` angelegt, die den Schlüssel als `Value` beinhaltet.

Value KeyGet(Key) Die Datei für eine Webanwendung namens `Key` wird ausgelesen und ihr Inhalt als `Value` zurückgegeben.

Der Nutzer bewahrt die Schlüssel auf einem USB-Stick auf und übergibt beim Starten des Browsers einen Parameter, in welchem Ordner der Schlüsselbund zu finden ist. Dort gibt es pro Webanwendung eine Datei, deren Inhalt der zu verwendende Schlüssel ist. Neben den Schlüsseln der Webanwendungen gibt es im KVStore Vergleichswerte für die Prüfsummen der JavaScript-Dateien. Diese sind für die Authentifizierung der Webanwendungen nötig, wie es im Abschnitt 5.4 noch beschrieben wird.

5.2 Browsererweiterung

Für die prototypische Implementierung wurde der Open-Source-Browser Chrome von Google [19] ausgewählt, der wie bereits in Abschnitt 3.4 beschrieben, über voneinander isolierte JavaScript-Engines verfügt.

Zur Verschlüsselung eignet sich die Verwendung einer existierenden Kryptographiebibliothek wie `Gcrypt`, die im Google Chrome Browser bereits vorhanden ist. Die Erweiterung des Browsers besteht darin, diese Verschlüsselungsfunktionalität der Webanwendung zur Verfügung zu stellen. Dafür wurde die Schnittstelle des DOM-Window-Objekts um die zwei Funktionen `enCrypt` und `deCrypt` ergänzt. Das Document Object Model (DOM) ist die Spezifikation der Schnittstelle, über die beispielsweise ein JavaScript auf das HTML-Dokument einer Webanwendung zugreift. Das DOM-Window-Objekt steht an oberster Stelle der Objekt-Hierarchie und repräsentiert ein Browser-Fenster oder einen Frame. Die Schnittstelle des DOM-Window-Objekts beinhaltet Methoden, wie beispielsweise `alert`, `prompt` oder `confirm`, mit denen ein JavaScript Warnmeldungen, Eingabeaufforderungen oder Bestätigungsdialoge öffnen kann.

In den Abbildungen 5.2 bis 5.5 ist die konkrete Erweiterung des DOM-Window-Objekts zu sehen.

```
void print();
void stop();

[Custom] DOMWindow open(in DOMString url,
                        in DOMString name,
                        in [Optional] DOMString options);

[Custom] DOMObject showModalDialog(in DOMString url,
                                   in [Optional] DOMObject dialogArgs,
                                   in [Optional] DOMString featureArgs);

void alert(in DOMString message);
                                        // Erweiterung der Schnittstelle des
DOMString enCrypt(in DOMString mKey);   // Window-Objekts um eine Verschlüsselungs-
DOMString deCrypt(in DOMString mKey);   // und eine Entschlüsselungsfunktion

boolean confirm(in DOMString message);
```

Abbildung 5.2: Die Schnittstelle des DOM-Window-Objekts mit der Erweiterung

```
        void focus();
        void blur();
        void close();
        void print();
        void stop();

        void alert(const String& message);
                                            // Erweiterung des Window-Objekts
        String enCrypt(String mKey);        // um eine Verschlüsselungs- und
        String deCrypt(String mKey);        // eine Entschlüsselungsfunktion

        bool confirm(const String& message);
```

Abbildung 5.3: Deklaration der zusätzlichen Funktionen

```
1021 void DOMWindow::stop()
1022 {
1023     if (!m_frame)
1024         return;
1025     m_frame->loader()->stopForUserCancel(true);
1026 }
1027 String DOMWindow::enCrypt(String mKey)
1028 {
1029     if (!m_frame)
1030             return String();
1031     size_t inilength = 16;
1032     string etext = mKey.utf8().data();
1033     size_t txtLength =1 + etext.length();
1034     while ((txtLength % 16)!=0){
1035         txtLength++;}
1036     char * encrypted, *encrypted_base64;
1037     char * aesSymKey;
1038     char * e_iniVector= (char*)malloc(inilength); ;
1039     char * both = (char*)malloc(txtLength + inilength);
1040     char *appKey = (char*)malloc(inilength);
1041     char *key_base64, *loc;
1042     size_t kl,fs;
1043     String secOrigin = document()->securityOrigin()->toString();
1044     CString cloc= secOrigin.utf8();
1045     string location = cloc.data();
1046     location.erase(0,7);
1047     location.append(":key");
1048     loc = (char*)location.c_str();
1049     string searchKey = getV(loc);
1050     if(searchKey!="nichtvorhanden"){
1051         aesSymKey = (char*) g_base64_decode((const gchar*)searchKey.c_str(),&kl);
1052     }else{
1053         gcry_randomize(appKey, inilength, GCRY_VERY_STRONG_RANDOM);
1054         key_base64 = g_base64_encode((const guchar*) appKey, 16);
1055         putKV(loc, key_base64);
1056         aesSymKey = appKey;
1057     }
1058     gcry_create_nonce(e_iniVector, inilength);
1059     memcpy(both, (const char*) e_iniVector, inilength);
1060     encrypted = aesEncrypt(GCRY_CIPHER_MODE_CBC, e_iniVector, (char*) etext.c_str(), aesSymKey);
1061     memcpy(both + inilength, (const char*) encrypted, txtLength);
1062     encrypted_base64 = g_base64_encode((const guchar*) both, txtLength + inilength);
1063     free(encrypted);
1064     String r = encrypted_base64;
1065     g_free(encrypted_base64);
1066     free(both);
1067     free(e_iniVector);
1068     free(appKey);
1069     return r;
1070 }
1071 String DOMWindow::deCrypt(String mKey)
1072 {
```

Abbildung 5.4: Implementierung der Verschlüsselungsfunktion

Mit den zwei zusätzlichen Funktionen `enCrypt` und `deCrypt` kann die JavaScript-Engine V8 einen Text für eine Webanwendung ver- und entschlüsseln. Diese beiden Funktionen verwenden eine AES-Verschlüsselung im CBC-Modus mit einem zufälligen Initialisierungsvektor der Länge 16 Bytes und dem anwendungsspezifischen Schlüssel, der ebenfalls 16 Bytes lang ist. Diesen anwendungsspezifischen Schlüssel erhält die V8 über die Schnittstelle des KVStores. Diese Schnittstelle steht der V8 zur Verfügung, sie ist jedoch nicht Teil der Schnittstelle des

```
1068     free(appKey);
1069     return r;
1070 }
1071 String DOMWindow::deCrypt(String mKey)
1072 {
1073     string dtext = mKey.utf8().data();
1074     char * crypted_base64  = (char*) dtext.c_str();
1075     size_t length;
1076     size_t d_il =16;
1077     char * crypted = (char *)g_base64_decode((const gchar*)crypted_base64, &length);
1078     char * aesSymKey;
1079     char * d_iniVector = (char*)malloc(d_il);
1080     char * cryptedtext = crypted +d_il;
1081     char * d_appKey = (char*)malloc(d_il);
1082     char * d_key_base64, *d_loc;
1083     size_t d_kl,d_fs;
1084     String d_secOrigin = document()->securityOrigin()->toString();
1085     CString d_cloc= d_secOrigin.utf8();
1086     string d_location = d_cloc.data();
1087     d_location.erase(0,7);
1088     d_location.append(":key");
1089     d_loc = (char*) d_location.c_str();
1090     string d_searchKey = getV(d_loc);
1091 
1092     if(d_searchKey!="nichtvorhanden"){
1093         aesSymKey = (char*) g_base64_decode((const gchar*)d_searchKey.c_str(),&d_kl);
1094     }else{
1095         return mKey;
1096     }
1097     memcpy(d_iniVector, (const char*) crypted, d_il);
1098     char * text = aesDecrypt(GCRY_CIPHER_MODE_CBC, d_iniVector, cryptedtext, length-d_il, aesSymKey);
1099     g_free(crypted);
1100     String r = String::fromUTF8(text);
1101     free(text);
1102     free(d_iniVector);
1103     free(d_appKey);
1104     return r;
1105 }
1106 void DOMWindow::alert(const String& message)
1107 {
```

Abbildung 5.5: Implementierung der Entschlüsselungsfunktion

DOM-Window-Objekts, weil sonst jede Webanwendung Zugriff auf die Schlüssel von anderen Webanwendungen hätte und so die Daten von fremden Nutzern entschlüsseln könnte.

5.3 Instrumentierung der ToDo-Liste

Die ToDo-Liste von Sproutcore ist eine Webanwendung, in der sich der Nutzer eine Liste von Aufgaben erstellen kann (siehe Abbildung 5.6). Der Nutzer legt verschiedene Einträge an, indem er seine zu erledigenden Aufgaben benennt. Jeder Eintrag besitzt ein Statusfeld, das anzeigt, ob diese Aufgabe schon erledigt ist oder nicht. Die Aufgaben werden entsprechend ihres Status sortiert und können bei Bedarf gelöscht werden.

Die Einträge der ToDo-Liste werden beim Dienstanbieter in einer Datenbank abgelegt. Jeder Eintrag besitzt einen eindeutigen Index, einen Stringwert der die Beschreibung der Aufgabe enthält und ein boolesches Statusfeld, in dem der Status der Aufgabe gespeichert wird. Die Indizes der Einträge müssen unverschlüsselt bleiben, weil sie als Metadaten zur Identifizierung der Einträge dienen. Die Verschlüsselung aller drei Werte des Eintrages würde dazu führen, dass die Webanwendung nicht mehr funktioniert, weil keine Einträge identifiziert werden können. Der Text der Beschreibung kann verschlüsselt werden, wohingegen das Format des Statusfeldes als boolescher Wert in der Datenbank vorgegeben ist. Um dennoch den Originalstatus vor dem Dienstanbieter zu verbergen, wird bei der Verschlüsselung des Eintrages der Status zufällig gesetzt. Der ursprüngliche Status hingegen wird an die Beschreibung des Eintrages angehängt und mit ihr zusammen verschlüsselt.

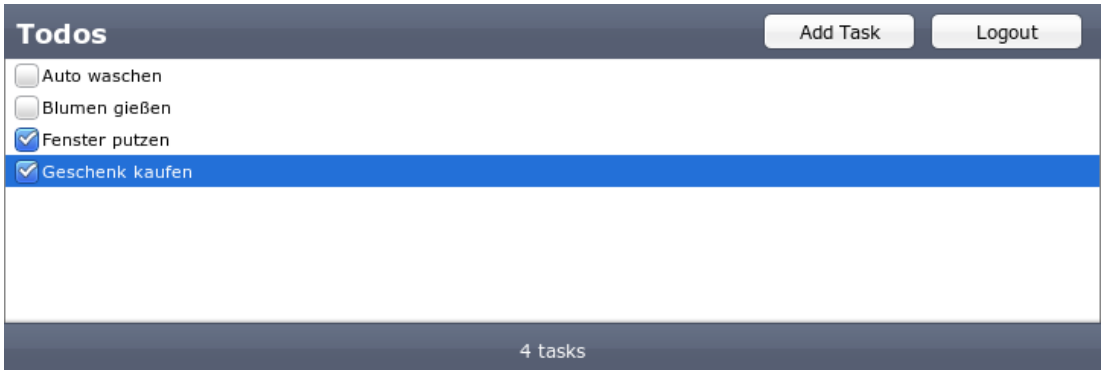

Abbildung 5.6: ToDo-Liste von Sproutcore

Eine Datenübertragung zwischen dem Browser und der Datenbank findet in den folgenden drei Fällen statt:

FETCH Wenn der Nutzer seine Webanwendung aufruft, werden alle Einträge, die ihm gehören, aus der Datenbank in den Browser geladen.

CREATE Bei einem neuen Eintrag wird die Bezeichnung und der Status an die Datenbank gesendet und unter einem neuen Indexwert abgelegt.

UPDATE Änderungen des Nutzers an der Bezeichnung oder dem Status werden zusammen mit dem vorhandenen Index gesendet.

Um Daten vor dem Versenden zu verschlüsseln, muss die Webanwendung instrumentiert werden. Hierbei wird Code eingefügt, der die von der Browsererweiterung zur Verfügung gestellten Verschlüsselungsfunktionen nutzt. Dafür wird bei jedem CREATE und jedem UPDATE der String der Bezeichnung verschlüsselt und der Status des Eintrages zufällig gesetzt. Der Originalstatus wird dabei zusammen mit der Bezeichnung verschlüsselt. Die Entschlüsselung aller Bezeichnungen und Wiederherstellung der Statusfelder erfolgt bei einem FETCH.

Als Beispiel wird in den nachfolgenden Auszügen aus dem JavaScriptcode der ToDo-Liste gezeigt, wie die UPDATE-Funktion instrumentiert wurde.

```
updateRecord: function(store, storeKey) {
  if (SC.kindOf(store.recordTypeFor(storeKey), Todos.Task)) {

      SC.Request.putUrl('/tasks/' + store.idFor(storeKey)).json()
      .notify(this, this.didUpdateTask, store, storeKey)
      .send(store.readDataHash(storeKey));
      return YES;

  } else return NO ;
},
```

Abbildung 5.7: UPDATE-Funktion ohne Instrumentierung

Die Abbildung 5.7 zeigt die ursprüngliche UPDATE-Funktion der ToDo-Liste, wie sie vom Sproutcore-Framework zur Verfügung gestellt wird.

In Abbildung 5.8 ist zu sehen, wie der Eintrag zunächst kopiert wird und dann anhand dieser Kopie ermittelt wird, welchen Wert der Status des Eintrages hat.

```
updateRecord: function(store, storeKey) {
  if (SC.kindOf(store.recordTypeFor(storeKey), Todos.Task)) {

      var udes= SC.clone(store.readDataHash(storeKey));        // Erzeugung einer Eintragskopie

      if(udes.isDone==true)                                    // Der Status des Eintrages
              udes.description = (udes.description + 'getan'); // wird ermittelt und
      if(udes.isDone==false)                                   // der Beschreibung des
              udes.description = (udes.description + 'zutun'); // Eintrages hinzugefügt.

      var uzufall =  Math.random();                            // Mit Hilfe einer Zufallszahl
      var ustat = 2 * uzufall;                                 // wird der Orginalstatus
      if(ustat <= 1)                                           // des Eintrages verborgen.
              udes.isDone=false;                               // Ein zufälliger boolescher
      else                                                     // Wert wird als Status
              udes.isDone=true;                                // gesetzt.

      udes.description = enCrypt(udes.description);            // Verschlüsselung der Beschreibung

      SC.Request.putUrl('/tasks/' + store.idFor(storeKey)).json()
      .notify(this, this.didUpdateTask, store, storeKey)
      .send(udes);                                             // Senden des angepassten Eintrages
      return YES;

  } else return NO ;
},
```

Abbildung 5.8: UPDATE-Funktion mit Instrumentierung

Dieser Originalstatus wird in Textform zu der Beschreibung des Eintrages hinzugefügt. Danach erhält der Eintrag einen zufälligen Status und die Beschreibung des Eintrages wird verschlüsselt. Dieser verschlüsselte Eintrag wird dann an die Server-Seite der Webanwendung übertragen.

5.4 Authentifizierung der ToDo-Liste

Wie in Abschnitt 4.4.2 beschrieben, muss nach dem Herunterladen des JavaScripts dessen Korrektheit verifiziert werden. Dies geschieht über die Prüfung eines Zertifikats, das von einer vertrauenswürdigen dritten Instanz ausgestellt wurde. Um diesen Vorgang zu simulieren, wurden in der prototypischen Implementierung die JavaScript-Bestandteile der Webanwendung mit einer MD5-Prüfsumme versehen.

Der JavaScriptcode der ToDo-Liste setzt sich aus fünf einzelnen JavaScripts zusammen, die alle separat mit einer MD5-Prüfsumme versehen werden. Diese wird als Kommentarzeile an das jeweilige JavaScript angehängt.

In Abbildung 5.9 ist zu sehen, dass die HTML-Datei der ToDo-Liste fünf separate JavaScript-Bestandteile beinhaltet. Drei Bestandteile des JavaScriptcodes sind direkt in der HTML-Datei aufgeführt (ihre MD5-Prüfsummen sind blau unterstrichen) und zwei JavaScript-Dateien werden nachgeladen (grün unterstrichen), das JavaScript des Sproutcore-Frameworks und das JavaScript der Webanwendung.

Stellvertretend für die zwei nachgeladenen JavaScript-Dateien ist in der Abbildung 5.10 zu sehen, wie die MD5-Prüfsumme als Kommentar zu dem JavaScript der ToDo-Liste hinzugefügt wurde.

Damit diese Prüfsummen automatisch an die vom Sproutcore-Framework erzeugten JavaScript-Dateien angehängt werden, wurde das in der Abbildung 5.11 zu

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=8" />
    <meta http-equiv="Content-Script-Type" content="text/javascript" />
    <title>Todos</title>
  <script type="text/javascript">var SC=SC||{BUNDLE_INFO:{},LAZY_INSTANTIATION:{}};SC.browser=(function(){var c=navigator.userAgent
                                .toLowerCase();var a=(c.match(/.+(?:rv|it|ra|ie)[\/: ]([\d.]+)/)||[])[1];var b={version:a,safari:(/webkit/)
                                .test(c)?a:0,opera:(/opera/).test(c)?a:0,msie:(/msie/).test(c)&&!(/opera/).test(c)?a:0,mozilla:(/mozilla/)
                                .test(c)&&!(/(compatible|webkit)/).test(c)?a:0,mobileSafari:(/apple.*mobile.*safari/)
                                .test(c)?a:0,windows:!!(/(windows)/).test(c),mac:!!((/(macintosh)/).test(c)||(/(mac os x)/)
                                .test(c)),language:(navigator.language||navigator.browserLanguage).split("-",1)[0]};b.current=b.msie?"msie":b
                                .mozilla?"mozilla":b.safari?"safari":b.opera?"opera":"unknown";return b})();
                                SC.bundleDidLoad=function(a){var b=this.BUNDLE_INFO[a];if(!b){b=this.BUNDLE_INFO[a]={}}b.loaded=true};
                                SC.bundleIsLoaded=function(a){var b=this.BUNDLE_INFO[a];return b?!!b.loaded:false};
                                SC.loadBundle=function(){throw"SC.loadBundle(): SproutCore is not loaded."};
                                SC.setupBodyClassNames=function(){var e=document.body;if(!e){return}var c,a,f,b,g,d;c=SC.browser.current;
                                a=SC.browser.windows?"windows":SC.browser.mac?"mac":"other-platform";d=document.documentElement.style;
                                f=(d.MozBoxShadow!==undefined)||(d.webkitBoxShadow!==undefined)||(d.oBoxShadow!==undefined)||
                                (d.boxShadow!==undefined);b=(d.MozBorderRadius!==undefined)||(d.webkitBorderRadius!==undefined)||
                                (d.oBorderRadius!==undefined)||(d.borderRadius!==undefined);g=e.className?e.className.split(" "):[];
                                if(f){g.push("box-shadow")}if(b){g.push("border-rad")}g.push(c);g.push(a);if(SC.browser.msie==7){g.push("ie7")}
                                if(SC.browser.mobileSafari){g.push("mobile-safari")}e.className=g.join(" ")};
                                if((typeof SC!=="undefined")&&SC&&SC.bundleDidLoad){SC.bundleDidLoad("sproutcore/bootstrap")};
                                /*MD5CS:e58f2ce4e29557e9545b93bc1e2d4f89*/</script>

  <link href="/static/sproutcore/en/8bb8d72df8d2cd7e336defa7c14db956a548f7b6/stylesheet-packed.css" rel="stylesheet" type="text/css" />
  <link href="/static/sproutcore/standard_theme/en/e8710e7b5a56381dc244c10d33fc9dc64c4b3e86/stylesheet.css" rel="stylesheet" type="text/css" />


  </head>

  <body class="sc-theme focus" style="overflow:hidden;" >
    <script type="text/javascript">if(SC.setupBodyClassNames){SC.setupBodyClassNames()};/*MD5CS:f08c3e4f6645c37ee47a0360e6a56601*/</script>


    <div id="loading">
      <p class="loading">Loading...<p>
    </div>

    <script type="text/javascript" src="/static/sproutcore/en/8bb8d72df8d2cd7e336defa7c14db956a548f7b6/javascript-packed.js"></script>
    <script type="text/javascript" src="/static/todos/en/30c6cf7c0ea62fba41fdf25a14a3f28bdb17aa3c/javascript.js"></script>
    <script type="text/javascript">String.preferredLanguage = "en";/*MD5CS:2755cdbcaecd56400bc47a5359bdb510*/</script>
  </body>
</html>
```

Abbildung 5.9: JavaScript-Bestandteile mit der jeweiligen MD5-Prüfsumme

```
...
    bottomView: SC.ToolbarView.design({
      layout: { bottom: 0, left: 0, right: 0, height: 32 },
      childViews: 'summaryView'.w(),
      anchorLocation: SC.ANCHOR_BOTTOM,

      summaryView: SC.LabelView.design({
        layout: { centerY: 0, height: 18, left: 20, right: 20 },
        textAlign: SC.ALIGN_CENTER,
        valueBinding: "Todos.tasksController.summary",
      })
    })
  })

});

Todos.main = function main() {

  Todos.getPath('mainPage.mainPane').append() ;
  var tasks = Todos.store.find(Todos.TASKS_QUERY);
  Todos.tasksController.set('content', tasks);
};

function main()
{
        Todos.main();
}
/*MD5CS:35809b353b25dfbb235a99f2f60b95c3*/
```

Abbildung 5.10: JavaScript der ToDo-Liste mit angehängter MD5-Prüfsumme

sehende Ruby-Skript, das die JavaScript-Dateien für die Sproutcore-Anwendungen

zusammensetzt, um vier Zeilen ergänzt, die an das jeweilige JavaScript die entsprechende MD5-Prüfsumme in einer Kommentarzeile anhängen.

```
  class Builder::Combine < Builder::Base

    def build(dst_path)
      lines = []
      entries = entry.ordered_entries || entry.source_entries

      target_name = entry.target.target_name.to_s.sub(/^\//,'')
      if entry.top_level_lazy_instantiation && entry.combined
        lines << "
if ((typeof SC !== 'undefined') && SC && !SC.LAZY_INSTANTIATION) {
  SC.LAZY_INSTANTIATION = {};
}
if(!SC.LAZY_INSTANTIATION['#{target_name}']) {
  SC.LAZY_INSTANTIATION['#{target_name}'] = [];
}
SC.LAZY_INSTANTIATION['#{target_name}'].push(
  (
    function() {
"
      end

      entries.each do |entry|
        src_path = entry.stage!.staging_path
        next unless File.exist?(src_path)
        lines << "/* >>>>>>>>>> BEGIN #{entry.filename} */\n"
        lines += readlines(src_path)
      end
      if entry.top_level_lazy_instantiation && entry.combined
                lines << "
    }
  )
);
"
      end

      if target_name == "todos"                          # Wenn JavaScripts für die ToDo-Liste
        digest = Digest::MD5.hexdigest(lines.join(""))   # erzeugt werden, wird die MD5-
        lines << "/*MD5CS:#{digest}*/"                   # Prüfsumme des JavaScripts am Ende
      end                                                # als Kommentarzeile hinzugefügt.

      writelines dst_path, lines
    end

  end
```

Abbildung 5.11: JavaScript-Dateien werden mit einer MD5-Prüfsumme versehen

Die korrekten MD5-Prüfsummen aller fünf JavaScript-Bestandteile werden im KVStore als Vergleichswerte abgelegt. Der Browser wurde so erweitert, dass er vor der Ausführung der Webanwendung überprüft, ob die angehängte MD5-Prüfsumme dem jeweiligen JavaScriptcode entspricht und ob sie identisch mit dem entsprechenden Vergleichswert ist. Wenn die Webanwendung nicht als korrekt verifiziert werden kann, wird statt dem JavaScript eine Warnmeldung ausgeführt. Somit wird der Nutzer davor bewahrt, eine manipulierte Webanwendung auszuführen.

Des Weiteren wird vor der Ausführung des Browsers die HOME-Variable auf den USB-Stick umgeleitet, so dass der Browser den KVStore immer in demselben Ordner vorfindet und neu generierte Schlüssel dort abspeichern kann. Damit hat der Nutzer auch die Einstellungen in seinem Browser, wie zum Beispiel seine Bookmarks, immer parat und kann sie an einem beliebigen Rechner aufrufen. Diese mobile Nutzung von Programmen und Einstellungen wird auch bei dem von

Microsoft und SanDisk entwickelten U3-System, beziehungsweise seinem Nachfolger StartKey, verwendet [20]. Der Nutzer kann bei diesem Verfahren an einem beliebigen Windows-Rechner über seinen USB-Stick oder seine SD-Speicherkarte Anwendungen starten, ohne sie vorher auf dem Rechner installieren zu müssen und ohne dass seine privaten Daten auf dem Rechner verbleiben, sobald er den USB-Stick oder die SD-Speicherkarte wieder entfernt.

6 Bewertung

Dieses Kapitel gibt einen Einblick in welcher Form der Datenschutz einer Webanwendung verbessert werden konnte. Darauf folgt eine Beschreibung des Aufwandes, den der Dienstanbieter leisten muss, um seinen Nutzern eine Webanwendung mit verbessertem Datenschutz bieten zu können. Danach wird mit Hilfe von Messungen gezeigt, ob der Einsatz des verbesserten Datenschutzes eine Auswirkung auf die Ausführungszeit der Webanwendung hat.

6.1 Erreichte Verbesserung des Datenschutzes

Zu Beginn werden die beiden unterschiedlichen Perspektiven auf die Daten des Nutzers gezeigt. Darauf folgt die Darstellung des Szenarios, in dem ein Nutzer ohne die erforderlichen Schlüssel auf einem USB-Stick versucht, die Webanwendung auszuführen. Anschließend wird gezeigt, wie der Nutzer daran gehindert wird, ein JavaScript auszuführen, das manipuliert wurde. Zum Abschluss ist zu sehen, wie eine Webanwendung scheitert, die versucht die Daten aus fremden Webanwendungen zu entschlüsseln.

6.1.1 ToDo-Liste aus der Sicht des Nutzers und des Dienstanbieters

Die Webanwendung wird zunächst aus der Sicht des Nutzers gezeigt und danach aus der Sicht des Dienstanbieters, der in seiner Datenbank die Daten des Nutzers nur in verschlüsselter Form sehen kann. Wie in Abschnitt 4.5.1 beschrieben, ermöglicht der USB-Stick mit den Schlüsseln für die Webanwendungen mit verbessertem Datenschutz dem Nutzer eine mobile Nutzung seiner Webanwendungen von einem beliebigen Rechner aus, solange dieser ihm einen erweiterten Browser zur Verfügung stellt.

In Abbildung 6.1 ist die ToDo-Liste dargestellt, so wie der Nutzer sie sieht und man kann in diesem Beispiel sehen, dass der Nutzer sieben Einträge eingegeben hat, von denen drei als „erledigt“ markiert wurden. Alle Aufgaben werden dem Nutzer sortiert nach Status und in alphabetischer Reihenfolge angezeigt.

Entsprechend dazu wird in Abbildung 6.2 dargestellt, wie der Dienstanbieter, der nicht in Besitz des Schlüssel ist, die Daten in seiner Datenbank vorfindet. Es werden für ihn ebenfalls sieben Einträge angezeigt. Jedoch werden nur zwei Einträge als „erledigt“ aufgeführt und die Beschreibungen aller Eintrage sind verschlüsselt. Zudem erkennt der Dienstanbieter auch nicht die Reihenfolge der Einträge, wie sie dem Nutzer angezeigt werden.

Durch diesen Vergleich wird deutlich, dass der Dienstanbieter die Daten in seiner Datenbank zwar verwalten kann, aber keinen Schlüssel zur Entschlüsselung besitzt und somit keinen Zugriff auf den Inhalt der Daten hat.

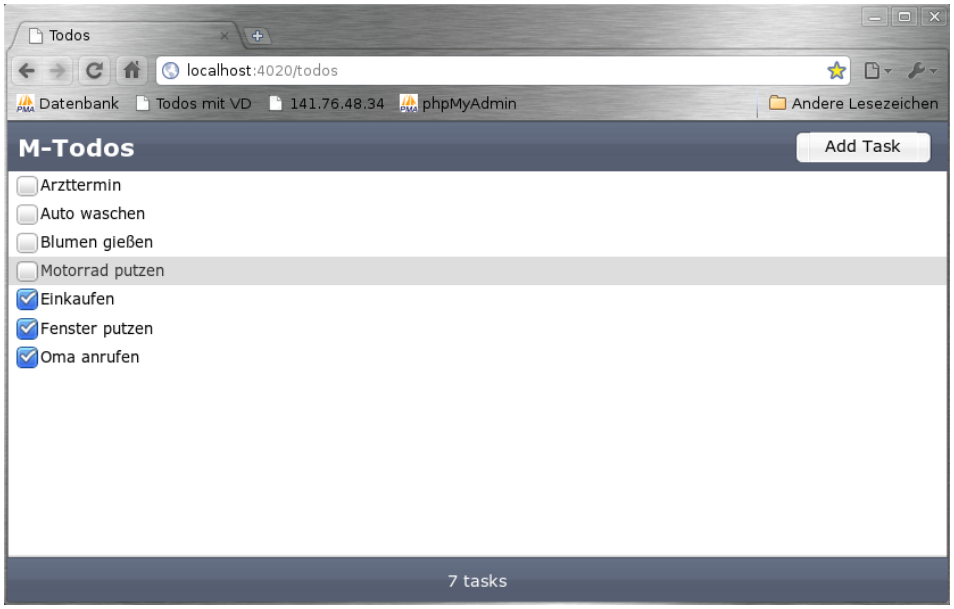

Abbildung 6.1: ToDo-Liste aus der Sicht des Nutzers

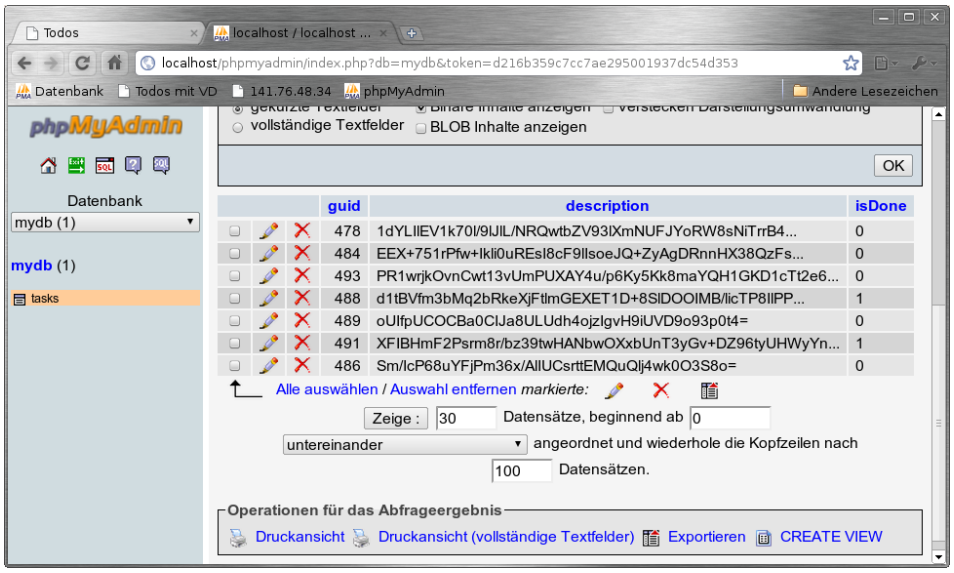

Abbildung 6.2: ToDo-Liste aus der Sicht des Dienstanbieters

Es wird allerdings auch deutlich, dass durch das Verbergen des Originalstatus, gewisse Funktionalitäten eingeschränkt sein können. Wenn man zum Beispiel nur die Teilmenge der als „erledigt“ markierten Einträge abrufen wollte, würden hier die falschen Einträge ausgewählt werden. Um diese Funktionalität trotz Verschlüsselung realisieren zu können, müssten alle Einträge abgerufen und entschlüsselt werden, denn nur so kann der korrekte Status der Einträge ermittelt werden.

6.1.2 Ausführung ohne Berechtigung

Wenn ein Nutzer versucht, die Webanwendung auszuführen, ohne dass er in Besitz des USB-Sticks mit dem Schlüssel ist, erhält er die in der Abbildung 6.3 zu sehende Warnmeldung. Nur der Nutzer, der den USB-Stick mit dem richtigen Schlüssel besitzt, erhält Zugriff auf seine persönlichen Daten.

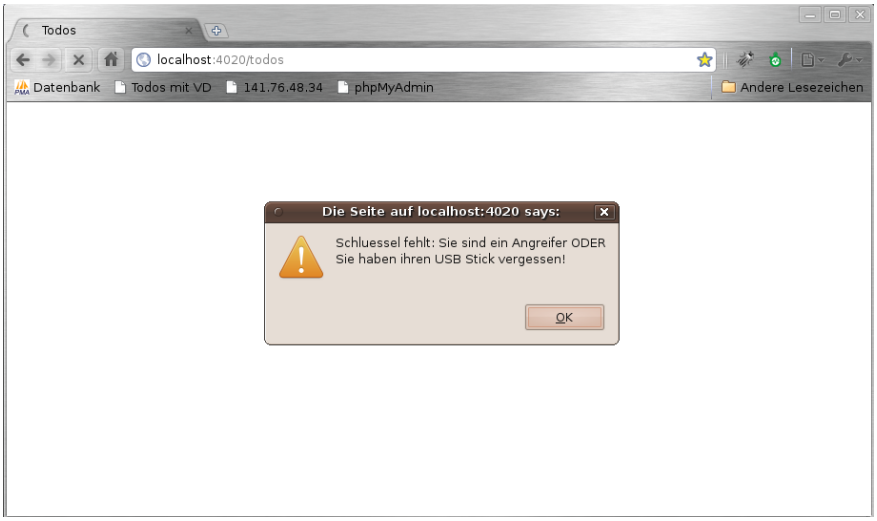

Abbildung 6.3: Fehlender Schlüssel

6.1.3 Manipuliertes JavaScript

Wie bei der Verschlüsselung verlässt sich der Nutzer auf seinen Browser im Hinblick auf die Authentifizierung des JavaScripts der Webanwendung.

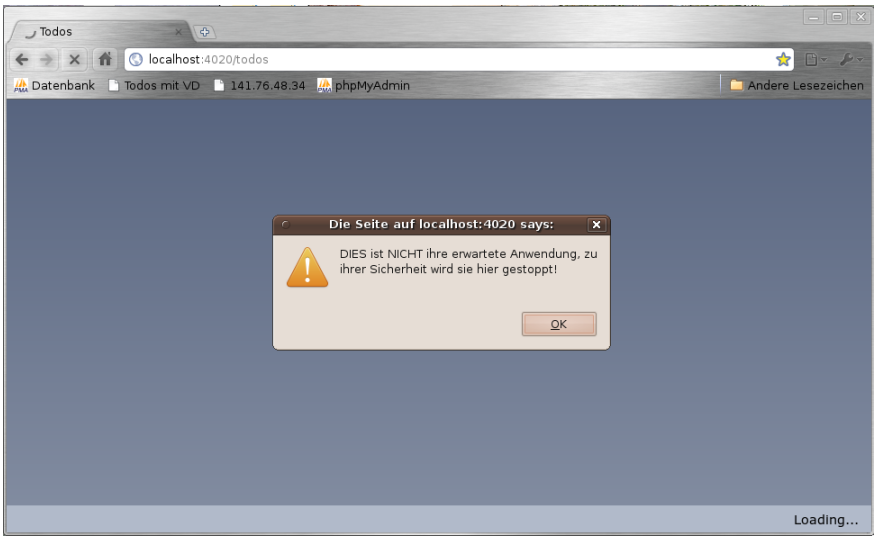

Abbildung 6.4: Manipuliertes JavaScript

Bevor eine Webanwendung mit verbessertem Datenschutz ausgeführt wird, überprüft der Browser, ob das beiliegende Zertifikat gültig ist. Sollte das nicht der Fall sein, warnt der Browser den Nutzer, wie es in der Abbildung 6.4 zu sehen ist, und hindert ihn daran, eine offensichtlich manipulierte Webanwendung auszuführen.

6.1.4 Entschlüsselungsversuch einer anderen Webanwendung

In der Abbildung 5.4 sieht man, wie auf der JavaScript-Konsole ein beispielhafter Text verschlüsselt wird und die Entschlüsselung auf derselben Seite Erfolg hat.

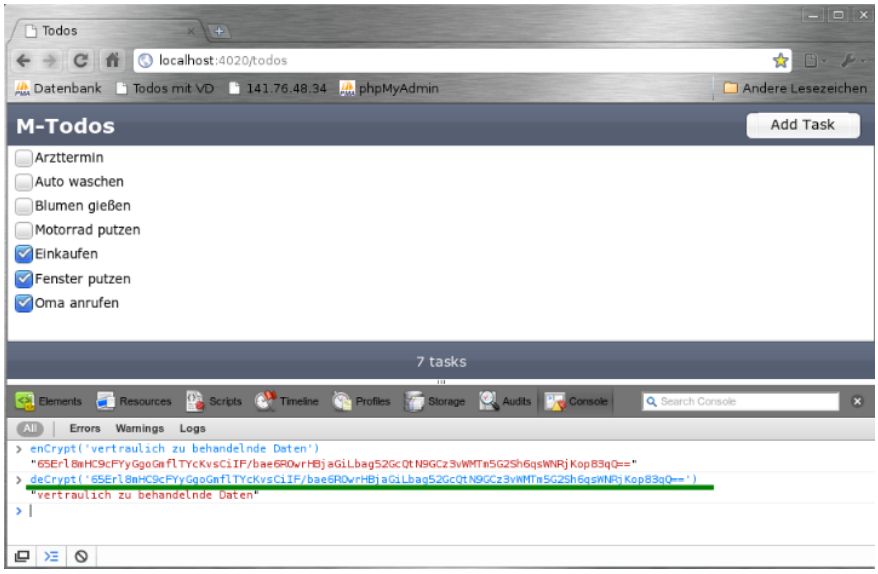

Abbildung 6.5: Verschlüsselung der Daten durch eine Webanwendung

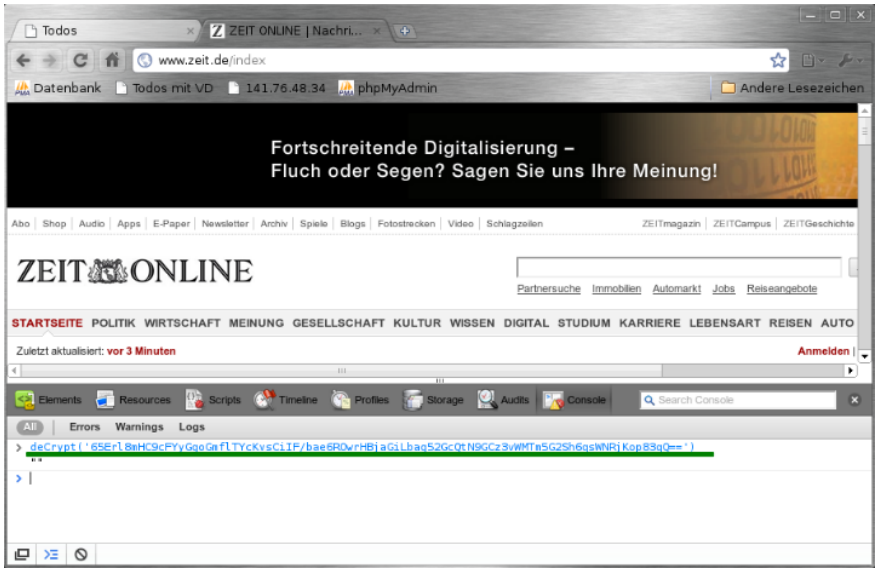

Abbildung 6.6: Versuch einer anderen Webanwendung, die Daten zu entschlüsseln

Sobald jedoch eine andere Webanwendung versucht, genau denselben verschlüsselten Text (grün unterstrichen) zu entschlüsseln, schlägt dies fehl und sie erhält keinen entschlüsselten Text, wie es in der Abbildung 6.6 zu sehen ist.

Dies zeigt, dass die Webanwendungen keinen Einfluss darauf haben, welcher Schlüssel bei der Ver- und Entschlüsselung verwendet wird. Der Browser identifiziert die Webanwendung genauso wie bei der Durchsetzung der Same-Origin-Policy. In Abhängigkeit der auf diesem Weg erhaltenen Bezeichnung der Webanwendung wird der Schlüssel aus dem KVStore ausgewählt.

6.2 Aufwand

Wie schon im Abschnitt 4.3 beschrieben, muss die Webanwendung vom Dienstanbieter instrumentiert werden, so dass die persönlichen Daten des Nutzers in der Datenbank auf der Server-Seite nicht im Klartext vorliegen.

Der Dienstanbieter muss in der Webanwendung zunächst alle HTTP-Anfragen identifizieren und bei jeder einzelnen bestimmen, ob sie dazu dient Daten zur Server-Seite oder zur Client-Seite zu übertragen. Dementsprechend muss vor der Übertragung eine Ver- oder Entschlüsselung stattfinden.

Im nächsten Schritt wird das Format der zu übertragenden Daten untersucht, um festzustellen, welche Bestandteile unverschlüsselbare Metadaten sind und welche Nutzerdaten ohne Weiteres verschlüsselt werden können. Als letztes muss sich der Dienstanbieter überlegen, wie er strukturgebundene Informationen so verschleiern kann, dass sie ihre Funktionalität nicht verlieren und trotzdem den Inhalt der Daten nicht preisgeben. Dieser Teil der Instrumentierung ist wahrscheinlich der aufwendigste, weil für jede strukturgebundene Information eine individuelle Strategie entworfen werden muss, die sowohl die Funktionalität erhält aber auch das Verbergen des Inhalts der Daten realisiert.

Bei der prototypischen Implementierung in dieser Arbeit wurde die ToDo-Liste so instrumentiert, dass die strukturgebundene Information des Status eines Eintrages vor der Übertragung zur Server-Seite zufällig gesetzt wird und der Originalwert zusammen mit der Beschreibung des Eintrages verschlüsselt wird.

In der Tabelle 6.1 ist zu sehen, wie viele Zeilen JavaScriptcode dem ursprünglichen JavaScript der ToDo-Liste hinzugefügt wurden, um die Instrumentierungen zu realisieren.

	SLOC	%
ToDo-Liste im Original	221	
ToDo-Liste mit Instrumentierung	260	
Anteil der Instrumentierung	39	17,6

Tabelle 6.1: Codekomplexität der Instrumentierungen

Diese Werte wurden mit CLOC [21] ermittelt und in Source Lines of Code (SLOC) angegeben. Es wird deutlich, dass die Instrumentierungen das JavaScript der ToDo-Liste um 17,6 % vergrößern, was zunächst viel erscheint. Es ist jedoch zu bedenken, dass die ToDo-Liste ein einfaches Demonstrationsbeispiel mit wenig JavaScriptcode ist, somit wirkt der Instrumentierungsanteil unverhältnismäßig groß.

	SLOC	%
Chrome Browser (Version 6.0.451.0) im Original	6.745.325	
Chrome Browser mit Erweiterungen	6.745.642	
Anteil der Erweiterungen	317	0,005
Größe des KVStores	84	0,001
Veränderungen insgesamt	401	0,006

Tabelle 6.2: Codekomplexität der Browsererweiterungen

Die Tabelle 6.2 zeigt, wie viele Zeilen Quellcode, angegeben in SLOC, dem Chrome Browser hinzugefügt wurden und wie groß der dazu gelinkte KVStore ist. Diese Daten wurden mit Hilfe von SLOCCount [22] von David A. Wheeler ermittelt. Dabei wird deutlich, dass der Anteil der Erweiterungen und des KVStores mit einem Prozentsatz von 0,006 verschwindend gering ist. Für einen verbesserten Datenschutz sind demnach im Browser nur minimale Änderungen vorzunehmen.

6.3 Messungen

Die durchgeführten Messungen sollen zeigen, inwieweit der Nutzer eine Verzögerung bemerkt, wenn er eine Webanwendung mit verbessertem Datenschutz ausführt. Dazu wurde die Dauer eines einzelnen Aufrufs der Ver- und Entschlüsselungsfunktionen mit verschieden langen Texten als Argument gemessen. Die darauffolgenden Messungen zeigen, wie lange es dauert, die ToDo-Liste im Browser zu laden. Diese Messungen wurden mit und ohne verbesserten Datenschutz durchgeführt und es wurde untersucht, ob es einen Unterschied es macht, wenn der Webserver und der Browser auf demselben Rechner ausgeführt werden oder auf zwei verschiedenen.

Bei der Ermittlung der Messwerte kamen zwei verschiedene Rechner zum Einsatz: Ein Dell Inspiron 630m, der über eine CPU-Leistung von 1,86 GHz und einen Arbeitsspeicher von 1 GB verfügt sowie ein Rechner der eine CPU-Leistung von 1,21 GHz und einen Arbeitsspeicher von 500 MB hat. Auf dem Inspiron wurden die Messungen durchgeführt, deren Ergebnisse in den Tabellen 6.3, 6.4 und 6.5 aufgelistet werden. Bei der Ermittlung der Messwerte, die in der Tabelle 6.6 zu sehen sind, wurde der Webserver auf dem Inspiron ausgeführt und der Browser auf dem zweiten Rechner.

6.3.1 Dauer einer Verschlüsselung und einer Entschlüsselung

Zunächst wurde mit einem beispielhaften JavaScript gemessen, wie lange eine Ver- und Entschlüsselung dauert, in Abhängigkeit von der Länge des zu transformierenden Textes. Wie in Abbildung 6.7 zu sehen ist, werden zufällige Texte mit einer Länge zwischen 100 und 10.000.000 Zeichen der Verschlüsselungsfunktion übergeben. Die Entschlüsselungsfunktion erhält die entsprechenden verschlüsselten Texte.

```
<script type ="text/javascript"><!--

var chars = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXTZabcdefghiklmnopqrstuvwxyz";
var string_length = 100;

for(string_length; string_length<= 10000000; string_length*=10){
        var randomstring = '';
        for (var i=0; i<string_length; i++){
                var rnum = Math.floor(Math.random() * chars.length);
                randomstring += chars.substring(rnum,rnum+1);
        }

        document.write("<br>");
        document.write("Zeichenlaenge: " +string_length);
        document.write("<br>");
        var vorher = new Date();
        var cipher = enCrypt(randomstring);
        var nachher = new Date();

        var one=nachher.getTime();
        var two=vorher.getTime();
        var diff = one - two;
        document.write("Dauer Verschluesselung: " +diff + "ms");
        document.write("<br>");

        var vor = new Date();
        var klartext = deCrypt(cipher);
        var nach =new Date();

        var three=nach.getTime();
        var four=vor.getTime();
        var differ = three - four;
        document.write("Dauer Entschluesselung: " +differ + "ms");
        document.write("<br>");
}

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

Abbildung 6.7: Zufällige Texte verschiedener Längen werden ver- und entschlüsselt

Die Messwerte in den Tabellen 6.3 und 6.4 zeigen, wie lange es dauert, Texte verschiedener Länge zu ver- und entschlüsseln. Die Messwerte, der Mittelwert (Ø) und die Standardabweichung (σ) wurden in Millisekunden angeben, dies entspricht der Messgenauigkeit bei Zeitmessungen innerhalb eines JavaScripts. Dabei war zu beobachten, dass erst ab einer Textlänge von 100.000 Zeichen (10^5) Werte im Millisekundenbereich ablesbar waren. Ab dieser Zeichenlänge erhöht sich bei einer Verlängerung des Textes um den Faktor 10 auch die Dauer der Ver- und Entschlüsselung ungefähr um den Faktor 10. Die Verschlüsselung eines Textes mit 10 Millionen Zeichen (10^7) dauerte demnach im Mittel 945,3 Millisekunden und die entsprechende Entschlüsselung 925,3 Millisekunden. Dies zeigt, dass selbst bei sehr großen Daten, die Ver- und Entschlüsselung weniger als eine Sekunde dauert.

Länge											Ø	σ
10^5	11	12	10	11	10	11	10	10	9	10	10,4	0,8
10^6	97	99	97	96	94	97	97	97	98	95	96,7	1,4
10^7	979	971	944	936	957	948	946	955	948	959	945,3	12,9

Tabelle 6.3: Dauer einer Verschlüsselung in Millisekunden

Länge											Ø	σ
10^5	9	8	11	9	8	9	8	9	9	8	8,8	0,9
10^6	94	94	92	92	95	93	91	92	92	93	92,8	1,2
10^7	941	930	910	909	932	920	922	932	921	936	925,3	10,7

Tabelle 6.4: Dauer einer Entschlüsselung in Millisekunden

6.3.2 Ladezeit der ToDo-Liste

Die folgenden Messungen zeigen, wie sich der verbesserte Datenschutz auf die Ladezeit der ToDo-Liste auswirkt. Dabei wurde der erweiterte Chrome Browser mit der URL der ToDo-Liste als Parameter aufgerufen. Dadurch wurden eventuelle Abweichungen vermieden, die zum Beispiel durch das Klicken auf den „Neu laden"-Button auftreten können. Bei der ersten Messreihe befinden sich der Webserver der Webanwendung und der Browser auf demselben Rechner. Die Ladezeit der ToDo-Liste wurde zuerst ohne verbesserten Datenschutz (~~VD~~) und danach mit verbessertem Datenschutz (VD) gemessen. Die Messwerte, der Mittelwert (Ø) und die Standardabweichung (σ) werden in Sekunden angegeben.

											Ø	σ
~~VD~~	0,86	1,31	0,89	0,84	0,91	0,95	0,88	0,89	0,94	1,14	0,96	0,14
VD	1,01	0,91	0,97	0,91	1,01	0,95	1,02	0,97	0,92	0,95	0,96	0,05

Tabelle 6.5: Ladezeiten in Sekunden, wenn sich der Webserver und der Browser auf demselben Rechner befinden

Wie in der Tabelle 6.5 zu sehen ist, hat der verbesserte Datenschutz keine verzögernde Auswirkung auf die Ladezeit der Webanwendung, wenn sowohl der Webserver als auch der Browser auf demselben Rechner ausgeführt werden.

Die zweite Messreihe zeigt die verschiedenen Ladezeiten der ToDo-Liste wenn der Webserver und der Browser auf verschiedenen Rechnern ausgeführt werden. Auch hier wurde zunächst die Ladezeit der ToDo-Liste ohne verbesserten Datenschutz (~~VD~~) gemessen und danach mit verbessertem Datenschutz (VD). Wie in der ersten Messreihe, werden die Messwerte, der Mittelwert (Ø) und die Standardabweichung (σ) in Sekunden angegeben.

In der Tabelle 6.6 wird deutlich, dass durch die Netzwerkverbindung zwischen Webserver und Browser eine Verzögerung um 1,76 Sekunden auftritt. Im Vergleich zur ersten Messreihe zeigen die Ergebnisse dieser Messreihe auch, dass der verbesserte Datenschutz im Mittel eine Verzögerung von 0,17 Sekunden verursacht. Dieser Wert liegt allerdings in der Größenordnung der Standardabweichung und ist somit vernachlässigbar.

											Ø	σ
~~VD~~	2,91	2,64	2,75	2,93	2,59	2,85	2,63	2,69	2,57	2,65	2,72	0,13
VD	2,86	2,79	2,94	2,77	2,84	2,84	2,89	2,95	3,08	2,94	2,89	0,09

Tabelle 6.6: Ladezeiten in Sekunden, wenn sich der Webserver und der Browser auf zwei verschiedenen Rechnern befinden

Die vorliegenden Messwerte zeigen, dass die Verbesserung des Datenschutzes in einer Webanwendung wie der ToDo-Liste, keine messbare Auswirkung auf die Ausführungszeit der Webanwendung hat. Es ist dabei unerheblich, ob der Webserver der Webanwendung und der Browser auf demselben oder auf zwei verschiedenen Rechnern ausgeführt werden. Eine Verzögerung durch den Einsatz des verbesserten Datenschutzes war in beiden Fällen nicht messbar. Mit dem verbesserten Datenschutz konnte also das Vertrauen reduziert werden, das der Nutzer dem Dienstanbieter einer Webanwendung entgegen bringen muss, ohne dass der Nutzer dafür eine längere Ausführungszeit in Kauf nehmen muss.

7 Ausblick und Zusammenfassung

Dieses Kapitel dient dazu einen Ausblick auf mögliche zukünftige Arbeiten zu geben und die vorliegende Arbeit mit ihren Ergebnissen zusammenzufassen.

7.1 Ausblick

In dieser Arbeit wurde anhand der ToDo-Liste beispielhaft gezeigt, wie eine Webanwendung des Sproutcore Frameworks durch den Einsatz des verbesserten Datenschutzes, den Nutzer davor bewahrt, seine persönlichen Daten dem Dienstanbieter preisgeben zu müssen.

Für eine zukünftige Arbeit ist es denkbar, diesen Vorteil jeder geeigneten Webanwendung des Sproutcore-Frameworks zur Verfügung zu stellen. Dafür müssen die speziell auf die ToDo-Liste angepassten Instrumentierungen verallgemeinert und in das Sproutcore-Framework eingefügt werden. Es ist zunächst zu ermitteln, welche Datenstruktur die zu übertragenden Daten haben, um zu unterscheiden, welche Metadaten unverschlüsselt bleiben müssen und welche Nutzerdaten verschlüsselt werden können. Besondere Aufmerksamkeit erfordert dabei die Verschleierung von strukturgebundenen Informationen. Diese besondere Art der Nutzerdaten hat einen festgelegten Wertebereich, wie beispielsweise einen booleschen Wert oder eine Uhrzeit. Dieses Format muss beim Verbergen des Inhaltes der Daten erhalten bleiben. Für diese Daten muss von Fall zu Fall untersucht werden, wie sich der Inhalt der Daten am besten verbergen lässt. Beispielsweise kann man, wie in dieser Arbeit gezeigt, den ursprünglichen Inhalt zusammen mit anderen textbasierten Nutzerdaten verschlüsseln und in der strukturgebundenen Information einen zufälligen Wert abspeichern.

7.2 Zusammenfassung

Das Ziel dieser Arbeit war es, das Vertrauen zu reduzieren, das der Nutzer dem Dienstanbieter einer Webanwendung entgegen bringen muss. Bisher hatten die Nutzer von Webanwendungen lediglich die Wahl, darauf zu vertrauen, dass der Dienstanbieter ihre persönlichen Daten vertraulich behandelt und sie nicht zweckentfremdet, oder darauf zu verzichten, die Webanwendung zu benutzen.

Mit der von mir vorgestellten Lösung, einem verbesserten Datenschutz für Webanwendungen, können Dienstanbieter ihren Nutzern Webanwendungen zur Verfügung stellen, die im Umgang mit den persönlichen Daten die Privatsphäre der Nutzer respektieren. Um dies realisieren zu können, muss der Dienstanbieter den client-seitigen Teil der Webanwendung so instrumentieren, dass die an ihn gesendeten Daten nur in verschlüsselter Form übertragen werden und dass nur der Nutzer, dem sie gehören, diese Daten in seinem Browser entschlüsseln kann. Die korrekte und vollständige Umsetzung dieser Instrumentierungen lässt sich der

Dienstanbieter von einer Zertifizierungstelle bestätigen. Dadurch muss sich der Nutzer nur auf das Urteil dieser vertrauenswürdigen dritten Instanz verlassen und nicht auf ein Versprechen des Dienstanbieters.

Für die Durchführung der Ver- und Entschlüsselung von Daten auf der Client-Seite ist es notwendig, dass der Browser des Nutzers um eine Verschlüsselungskomponente erweitert wird, mit der die Inhalte der persönlichen Daten des Nutzers vor dem Dienstanbieter verborgen werden. Zudem generiert der Browser dem Nutzer für jede Webanwendung einen Schlüssel und speichert diesen auf dem USB-Stick des Nutzers ab. So kann der Nutzer seine Webanwendungen ortsunabhängig an einem beliebigen Rechner mit erweitertem Browser aufrufen.

Anhand einer prototypischen Implementierung wurde in dieser Arbeit gezeigt, dass es möglich ist, Webanwendungen so zu gestalten, dass die persönlichen Daten der Nutzer vertraulich behandelt werden. Dafür wurden die ToDo-Liste des Sproutcore-Frameworks und der Chrome Browser von Google in der soeben beschriebenen Form instrumentiert und erweitert.

Es wurde mit Hilfe von Messungen ermittelt, ob der Browser durch den Einsatz des verbesserten Datenschutzes mehr Zeit zum Laden der Webanwendung braucht. Dies war nicht der Fall und somit wurde gezeigt, dass das notwendige Vertrauen, das der Nutzer dem Dienstanbieter einer Webanwendung entgegenbringen muss, durch den Einsatz des verbesserten Datenschutzes reduziert wird, ohne dass der Nutzer dafür Verzögerungen bei der Ausführung der Webanwendung hinnehmen muss.

Abbildungsverzeichnis

2.1 Modell einer Webanwendung 3
2.2 Angriffsmodell 5
2.3 Angriff auf die Datenübertragung 6
2.4 Angriff im Namen des Nutzers 6
2.5 Angriff durch den Dienstanbieter 7

4.1 Datenzugriff 19
4.2 Verwendung der Ver- und Entschlüsselung 23
4.3 Bedingte JavaScript-Ausführung 26

5.1 Implementierungskomponenten 29
5.2 Die Schnittstelle des DOM-Window-Objekts mit der Erweiterung . . . 30
5.3 Deklaration der zusätzlichen Funktionen 31
5.4 Implementierung der Verschlüsselungsfunktion 31
5.5 Implementierung der Entschlüsselungsfunktion 32
5.6 ToDo-Liste von Sproutcore 33
5.7 UPDATE-Funktion ohne Instrumentierung 33
5.8 UPDATE-Funktion mit Instrumentierung 34
5.9 JavaScript-Bestandteile mit der jeweiligen MD5-Prüfsumme 35
5.10 JavaScript der ToDo-Liste mit angehängter MD5-Prüfsumme 35
5.11 JavaScript-Dateien werden mit einer MD5-Prüfsumme versehen . . . 36

6.1 ToDo-Liste aus der Sicht des Nutzers 40
6.2 ToDo-Liste aus der Sicht des Dienstanbieters 40
6.3 Fehlender Schlüssel 41
6.4 Manipuliertes JavaScript 41
6.5 Verschlüsselung der Daten durch eine Webanwendung 42
6.6 Versuch einer anderen Webanwendung, die Daten zu entschlüsseln . 42
6.7 Zufällige Texte verschiedener Längen werden ver- und entschlüsselt . 45

Tabellenverzeichnis

6.1 Codekomplexität der Instrumentierungen 43
6.2 Codekomplexität der Browsererweiterungen 44
6.3 Dauer einer Verschlüsselung in Millisekunden 46
6.4 Dauer einer Entschlüsselung in Millisekunden 46
6.5 Ladezeiten in Sekunden, wenn sich der Webserver und der Browser auf demselben Rechner befinden 46
6.6 Ladezeiten in Sekunden, wenn sich der Webserver und der Browser auf zwei verschiedenen Rechnern befinden 47

Glossar

Add-On Eine optionale Erweiterung, die zusätzliche Funktionalität bereitstellt. S.5

AES Advanced Encryption Standard - Ein symmetrisches Kryptographiesystem, das den Klartext blockweise verschlüsselt. S.30

CBC Cipher Block Chaining - Eine Betriebsart von AES, die jeden Klartextblock vor der Verschlüsselung mit dem zuvor erzeugten Geheimtextblock per XOR verknüpft. S.30

Cookie Cookies sind kleine Dateien, die vom Webserver an den Browser gesendet werden und bei einem späteren Besuch dieser Webseite wieder an den Webserver geschickt werden. Sie können Passwörter, persönliche Einstellungen, wie beispielsweise einen Warenkorb oder auch Aufzeichnungen über das Surfverhalten des Nutzers beinhalten. S.9

DNS Domain Name System - Mit DNS wird die Namensauflösung im Internet realisiert. Eine für den Nutzer aussagekräftige URL, wie beispielsweise www.buchladen.de, wird aufgelöst zu einer eindeutigen IP-Adresse, wie zum Beispiel 209.85.149.106. S.3

HTML Hypertext Markup Language - Eine textbasierte Auszeichnungssprache mit der die Inhalte einer Webseite, wie Texte, Bilder oder Links, strukturiert werden. S.3

HTTP Hypertext Transfer Protocol - Ein Protokoll zur Übertragung von Daten über ein Netzwerk. Es wird hauptsächlich dazu eingesetzt, den Inhalt einer Webseite in den Browser zu laden. Aber auch nutzerspezifische Daten, die der Browser vom Webserver abruft oder an ihn sendet, werden mittels HTTP übertragen. S.3

HTTPS Hypertext Transfer Protocol Secure - Dieses Protokoll wird zur Verschlüsselung und Authentifizierung der Kommunikation zwischen Webserver und Browser eingesetzt. Es ermöglicht eine manipulations- und abhörsichere Übertragung von Daten. S.7

IP-Adresse Internet Protokoll Adresse - Jedes Gerät, das an ein lokales Netzwerk oder das Internet angeschlossen ist, besitzt eine eindeutig identifizierbare numerische Adresse. S.3

SQL-Injection Structured Query Language Injection - Der Begriff SQL-Einschleusung bezeichnet das Ausnutzen einer Sicherheitslücke im Zusammenhang mit SQL-Datenbanken, die durch mangelnde Maskierung oder Überprüfung von Metazeichen in Benutzereingaben

entsteht. Der Angreifer versucht auf diesem Weg, die Kontrolle über die Datenbank einer Webanwendung zu erlangen und die darin gespeicherten Daten auszuspionieren oder zu verändern. S.11

SSL Secure Socket Layer - Ein Verschlüsselungsprotokoll für die verfälschungs- und ausspähsichere Kommunikation im Internet. S.1

URL Uniform Resource Locator - Eine URL, im allgemeinen Sprachgebrauch auch als Internetadresse bekannt, ist die eindeutige Bezeichnung eines Webservers in einer für den Nutzer leicht merkbaren Form, wie beispielsweise www.buchladen.de . S.3

XSRF Cross-Site-Request-Forgery - Dieser Angriff bringt den Browser eines eingeloggten Nutzers dazu, eine manipulierte HTTP-Anfrage an eine verwundbare Webseite zu schicken. Dabei werden alle Authentifizierungsdaten des Nutzers mit übertragen. Das erlaubt es dem Angreifer, Anfragen an die Webanwendung zu senden, die diese für legitime Anfragen des Nutzers hält. S.9

XSS Cross-Site-Scripting - Beim XSS werden Daten von einem Nutzer an eine Webanwendung geschickt und diese dann an den Browser weitergeleitet, ohne dass eine Prüfung des Inhaltes stattfindet. Dadurch hat ein potentieller Angreifer die Möglichkeit, Skripte an einen Browser zu senden und einen schadhaften Code auf der Client-Seite ausführen zu lassen. S.9

Literaturverzeichnis

[1] DONGSEOK JANG, RANJIT JHALA, SORIN LERNER und HOVAV SHACHAM: An Empirical Study of Privacy-Violating Information Flows in JavaScript Web Applications. In: ANGELOS KEROMYTIS und VITALY SHMATIKOV (Herausgeber): CCS '10: Proceedings of the 17th ACM Conference on Computer and Communications Security. ACM Press, Oktober 2010.

[2] Alexa. `http://http://www.alexa.com`. Abruf: 08. Juni 2011.

[3] Firesheep. `http://codebutler.com/firesheep`. Abruf: 28. Mai 2011.

[4] TERRI ODA, GLENN WURSTER, P. C. VAN OORSCHOT und ANIL SOMAYAJI: SOMA: Mutual Approval for Included Content in Web Pages. In: CCS '08: Proceedings of the 15th ACM Conference on Computer and Communications Security, Seiten 89–98, New York, NY, USA, 2008. ACM.

[5] STEVEN CRITES, FRANCIS HSU und HAO CHEN: OMash: Enabling Secure Web Mashups via Object Abstractions. In: CCS '08: Proceedings of the 15th ACM Conference on Computer and Communications Security, Seiten 99–108, New York, NY, USA, 2008. ACM.

[6] DAVIDE BALZAROTTI, MARCO COVA, VIKTORIA V. FELMETSGER und GIOVANNI VIGNA: Multi-Module Vulnerability Analysis of Web-Based Applications. In: CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security, Seiten 25–35, New York, NY, USA, 2007. ACM.

[7] XIAONING DING, HAI HUANG, YAOPING RUAN, ANEES SHAIKH, BRIAN PETERSON und XIAODONG ZHANG: Splitter: A Proxy-based Approach for Post-Migration Testing of Web Applications. In: EuroSys '10: Proceedings of the 5th European Conference on Computer Systems, Seiten 97–110, New York, NY, USA, 2010. ACM.

[8] ALEXANDER YIP, NEHA NARULA, MAXWELL KROHN und ROBERT MORRIS: Privacy-Preserving Browser-Side Scripting With BFlow. In: EuroSys '09: Proceedings of the 4th ACM European Conference on Computer Systems, Seiten 233–246, New York, NY, USA, 2009. ACM.

[9] MICHAEL REIHER: Google Mail als sicheres E-mail-Archiv unter Benutzung eines Proxys, Januar 2009.

[10] STEPHEN CHONG, JED LIU, ANDREW C. MYERS, XIN QI, K. VIKRAM, LANTIAN ZHENG und XIN ZHENG: Secure Web Applications via Automatic Partitioning. In: SOSP '07: Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles, Seiten 31–44, New York, NY, USA, 2007. ACM.

[11] K. VIKRAM, ABHISHEK PRATEEK und BENJAMIN LIVSHITS: Ripley: Automatically Securing Web 2.0 Applications Through Replicated Execution. In: CCS '09: Proceedings of the 16th ACM Conference on Computer and Communications Security, Seiten 173–186, New York, NY, USA, 2009. ACM.

[12] CHARLES REIS und STEVEN D. GRIBBLE: Isolating Web Programs in Modern Browser architectures. In: EuroSys '09: Proceedings of the 4th ACM European Conference on Computer Systems, Seiten 219–232, New York, NY, USA, 2009. ACM.

[13] JINYUAN LI, MAXWELL KROHN, DAVID MAZIÈRES und DENNIS SHASHA: Secure Untrusted Data Repository (SUNDR). In: OSDI'04: Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation, Seiten 121–136, Berkeley, CA, USA, 2004. USENIX Association.

[14] GIUSEPPE ATENIESE, RANDAL BURNS, REZA CURTMOLA, JOSEPH HERRING, LEA KISSNER, ZACHARY PETERSON und DAWN SONG: Provable Data Possession at Untrusted Stores. In: CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security, Seiten 598–609, New York, NY, USA, 2007. ACM.

[15] PRINCE MAHAJAN, SRINATH SETTY, SANGMIN LEE, ALLEN CLEMENT, LORENZO ALVISI, MIKE DAHLIN und MICHAE WALFISH: Depot: Cloud storage with minimal trust. In: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10, Seiten 1–12, Berkeley, CA, USA, 2010. USENIX Association.

[16] Sproutcore Framework. `http://www.sproutcore.com` . Abruf: 17. Mai 2011.

[17] ToDo-Liste. `http://todos.demo.sproutcore.com` . Abruf: 17. Mai 2011.

[18] GEORGE C. NECULA: Proof-Carrying Code. In: Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '97, Seiten 106–119, New York, NY, USA, 1997. ACM.

[19] Google Chrome Browser. `http://dev.chromium.org/Home` . Abruf: 17. Mai 2011.

[20] MARY JO FOLEY: Microsoft preps StartKey: A 'Windows companion' on a USB stick. `http://www.zdnet.com/blog/microsoft/microsoft-preps-startkey-a-windows-companion-on-a-usb-stick/1232` , März 2008. Abruf: 12. Mai 2011.

[21] CLOC. `http://cloc.sourceforge.net` . Abruf: 17. Mai 2011.

[22] SLOCCount. `http://http://www.dwheeler.com/sloccount` . Abruf: 22. Mai 2011.