8 Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) schafft durch seinen ersten Namensbestandteil »Cross-Site« häufig eine unmittelbare Verbindung mit Cross-Site Scripting (XSS; Kapitel 7). Aber auch wenn beide Angriffe gelegentlich gemeinsam eingesetzt werden (Abschnitt 8.5), ist CSRF doch ein völlig anderer Angriff mit einem völlig anderen Ziel. Während bei XSS das Hauptaugenmerk des Angreifers auf den Benutzern Ihrer Webanwendung liegt, liegt es bei CSRF auf der Webanwendung selbst. Bei einem Cross-Site-Scripting-Angriff wird daher das »Vertrauen« eines Benutzers in eine Webanwendung ausgenutzt, bei einem Cross-Site-Request-Forgery-Angriff umgekehrt das »Vertrauen« der Webanwendung in einen Benutzer. Wichtig ist außerdem, dass bei einem CSRF-Angriff keine Daten gestohlen, sondern vielmehr Operationen (Requests) eines Angreifers im Namen eines ordnungsgemäß bei einer Webanwendung angemeldeten Benutzers ausgeführt werden. Immerhin ist es für den Angreifer nicht möglich, Daten abzufragen und diese an sich zurückliefern zu lassen.

8.1 Grundlagen

Cross-Site Request Forgery ist etwa seit dem Jahr 2000 ein Thema. Im Vergleich mit vielen anderen Angriffen sprechen wir damit von einem jüngeren und gleichzeitig deutlich unbekannteren Problem.[1] Sofern Sie Ihre Webanwendung nicht explizit davor schützen – beispielsweise durch die Verwendung eines Frameworks mit integriertem CSRF-Schutz oder durch eine der im Folgenden vorgestellten Gegenmaßnahmen – ist Ihre Webanwendung mit hoher Wahrscheinlichkeit verwundbar. Vereinfacht gesagt werden dabei Benutzer, die auf eine für den Angreifer interessante und verwundbare Webanwendung Zugriff haben, durch den Angreifer für den CSRF-Angriff instrumentalisiert.

[1] Bereits etwas länger bekannt ist allerdings die allgemeinere Form dieses Angriffs, der sogenannte *Confused Deputy*. Bei Cross-Site Request Forgery spricht man deshalb auch von einem Confused-Deputy-Angriff auf den Webbrowser, siehe: http://en.wikipedia.org/wiki/Confused_Deputy

Ein Angriff per Cross-Site Request Forgery macht sich grundsätzlich erst einmal den Umstand zunutze, dass alle Browser bei einem Request automatisch die zu dieser Domain gehörenden Zugangsdaten mit versenden. Darunter fallen Session-Informationen, wie beispielsweise die JSESSIONID in einem Session-Cookie. Unter normalen Umständen ist das ein gewünschtes und notwendiges Verhalten. Nur so kann eine über das zustandslose Hypertext Transfer Protocol (HTTP) angesprochene Webanwendung einen Benutzer wiedererkennen und Sessions verwenden. Ein heimlich von einem Angreifer über den Browser des Benutzers ausgelöster Request schickt diese Session-Informationen nun aber ebenso automatisch mit – ganz so, als ob der Benutzer selbst die Webanwendung aufruft und darin über Links navigiert oder Formulare ausfüllt. Auf der Serverseite hat eine gewöhnliche Webapplikation zunächst einmal keine Möglichkeit festzustellen, ob ein Benutzer eine Aktion wie das Abschicken eines Formulars oder das Löschen eines Datensatzes bewusst ausführen möchte oder ob der Request heimlich durch einen Angreifer initiiert wurde. Ganz allgemein ausgedrückt: Die Webanwendung überprüft, ob der Request vom Browser eines berechtigten (angemeldeten) Benutzers stammt, und nicht, ob der Request vom Benutzer selbst stammt.

Hinweis: CSRF im Online-Banking

Ein in Beschreibungen häufig verwendetes Beispiel für einen CSRF-Angriff ist eine Online-Banking-Webanwendung. Nur ist dieses Beispiel, zumindest in Deutschland, nicht wirklich realistisch. Schließlich ist nach der Anmeldung für fast jede weitere Aktion (Überweisung, Dauerauftrag usw.) noch eine separate Transaktionsnummer notwendig. Zwar verhindert eine TAN nicht den eigentlichen CSRF-Angriff, sie hält ihn normalerweise aber in der Ausführung auf. So wird eine unerwartete TAN-Abfrage vermutlich den meisten Anwendern seltsam genug vorkommen, um die Aktion abzubrechen.

Es sei denn, der Benutzer gibt eine Session-TAN an, wodurch alle Aktionen während dieser Anmeldung (der aktiven Session) ohne weitere TAN-Eingabe automatisch autorisiert sind. Damit wird Cross-Site Request Forgery ebenfalls im Online-Banking denkbar. So können beispielsweise Überweisungen nun ohne weitere TAN-Eingabe ausgeführt werden, und das Abschicken per CSRF ist – bei einer entsprechenden Verwundbarkeit der Banking-Webanwendung – damit ebenso denkbar. Da die Session-TAN-Funktionalität aber eher selten verwendet wird, ist das CSRF-Beispiel Online-Banking trotzdem vergleichsweise unrealistisch.

Wichtig ist an dieser Stelle, zu verstehen, dass der Angreifer keinesfalls den Benutzer direkt angreift und etwa in dessen Browser Aktionen ausführt oder gar (Schad-)Software auf dessen Computer installiert. Weiterhin wird kein Formular im Browser ausgefüllt, wie ein normaler Benutzer das tun würde. Das ist bei Angriffen mit CSRF weder notwendig noch gewünscht. Selbst die Benutzerverwaltung der Webanwendung wird nicht angegriffen. Stattdessen wartet der Angreifer, bis ein geeigneter Benutzer – idealerweise ein Administrator oder ein anderer Benutzer mit umfassenden Rechten in der Webanwendung – die von ihm manipulierte Webseite besucht. Der Angreifer nutzt anschließend den Browser des Benutzers zum Angriff mit heimlichen Requests auf die ausgewählte Webanwendung. Weder der Browser des Benutzers noch eine andere Software auf seinem Computer werden dabei in Mitleidenschaft gezogen.

Angreifer suchen kritische Formulare.

Weiterhin ist längst nicht jedes Formular für einen Cross-Site-Request-Forgery-Angriff interessant.[2] Simple Feedback- oder Kontaktformulare können unter bestimmten Voraussetzungen zwar ebenfalls ohne Ausfüllen des Formulars im Webbrowser abgeschickt werden, sind allerdings für einen Angreifer nicht weiter interessant, weil sie keine kritischen Backend-Operationen nach sich ziehen. Stattdessen sind die Angreifer auf der Suche nach »kritischen Formularen«, d. h. Formularen, die einen authentifizierten und autorisierten Benutzer erfordern und Operationen im Backend auslösen. Dies sind beispielsweise Bestellungen oder Verwaltungsaufgaben in einer Datenbank, wie etwa das Anlegen oder Bearbeiten von Stammdaten, Benutzern usw. Diese kritischen Formulare lösen zustandsverändernde Operationen (üblicherweise in der Datenbank) aus und müssen daher besonders geschützt werden.

Login-Formulare schützen

Geschützt werden müssen ferner Login-Formulare, selbst wenn hier noch kein authentifizierter Benutzer und damit keine vollständige Session vorhanden ist und keine kritischen Backend-Operationen nach dem Abschicken des Formulars ausgelöst werden. Mit einem CSRF-Schutz wird hier u. a. verhindert, dass ein Benutzer mit untergeschobenen Zugangsdaten angemeldet wird und ein Angreifer später die mit diesem Account durchgeführten Aktionen ausspioniert.

Logout-Formulare schützen

Logout-Formulare müssen gleichermaßen geschützt werden. Der Schutz soll hier beispielsweise verhindern, dass Anwender abgemeldet werden, sich dann per manipuliertem Formular wieder anmelden und dabei einem Angreifer ihre Zugangsdaten mitteilen.

[2] Wie Sie im Laufe des Kapitels sehen werden, sind Formulare für CSRF-Angriffe ohnehin nicht das interessanteste Ziel: Links können wesentlich einfacher ausgenutzt werden.

Natürlich können Sie einfach alle Formulare mit den in den folgenden Abschnitten beschriebenen Maßnahmen sichern. Notwendig und empfehlenswert ist das allerdings nicht. Identifizieren Sie die kritischen Operationen (in Formularen und Links), und statten Sie nur diese mit einem CSRF-Schutz aus. Die dadurch gesparte Zeit investieren Sie besser in andere Schutzmaßnahmen.

Als Einstiegsbeispiel ins Thema CSRF dient im Folgenden eine Webanwendung mit integrierter Benutzer- bzw. Kundenverwaltung. Diese besitzt für Administratoren eine einfache Übersicht, in der diese alle Benutzer verwalten und über Links den Status oder das Bestelllimit ändern bzw. über einen Button Benutzer löschen können (Abbildung 8-1).

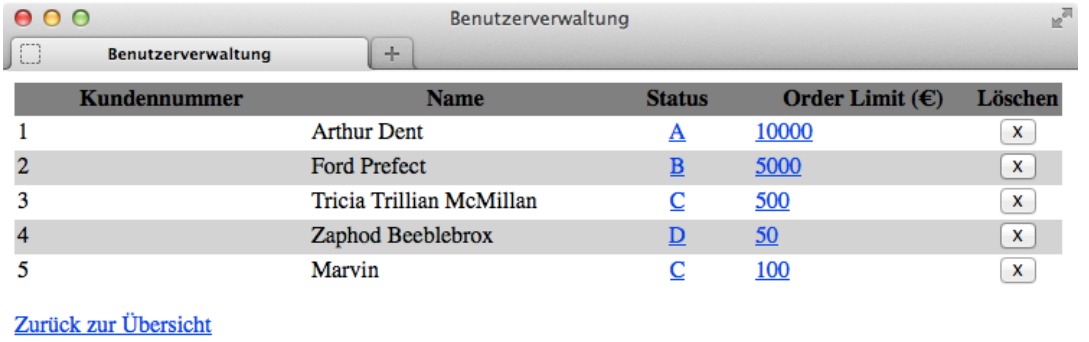

Abb. 8-1
Benutzerverwaltung in einer Webapplikation

In der Übersicht in Abbildung 8-1 ist kein Formular vorhanden. Erst die hinter den Links zur Status- und Limit-Änderung liegenden Seiten enthalten einfache Formulare. Operationen, die direkt über einen Link oder Button per `GET`-Request aufgerufen werden, sind für einen Angreifer sehr viel einfacher ausnutzbar als umfangreiche und per `POST`-Request versandte Formulare. Links funktionieren direkt mit URL-Parametern und damit ohne das Ausfüllen eines Formulars. Beispiele dafür sind das Löschen von Benutzern oder die Änderung der Benutzerrolle, etwa in eine Administratorgruppe. Solche Links werden sehr viel häufiger für Cross-Site-Request-Forgery-Angriffe verwendet als Formulare.

Ein solchermaßen funktionsgeladener Link sieht beispielsweise wie folgt aus:

```
http://www.site.com/stocks?buy=1000&stock=abc
```

Der Link enthält die durchzuführende Operation (`buy`), die Menge (`1000`), den Gegenstand der Operation (`stock`) und einen Identifikator (`abc`). Bei einer normalen Verwendung klickt der dazu berechtigte Benutzer auf diesen Link und löst die Kaufoperation aus.

Anstatt die Webapplikation selbst direkt anzugreifen, wählt der Angreifer ein anderes und indirekteres Vorgehen. Dazu präpariert er eine öffentliche Webseite oder verfasst einen Beitrag in einem Forum, das häufiger von Benutzern und möglicherweise auch vom Administrator der anzugreifenden Webapplikation besucht wird. Um die Wahrscheinlichkeit eines Besuchs zu erhöhen, verwendet der Angreifer dafür ein Benutzerforum der als Ziel ausgewählten Webanwendung und verfasst einige Beiträge zu bekannten Problemen.

In einen seiner dort verfassten Beiträge bettet er neben seinem ungefährlichen (normalen) Text beispielsweise den folgenden kurzen HTML-Code ein (Listing 8-1). Das `src`-Attribut verweist dabei nicht auf ein existierendes Bild, sondern auf die auszuführende Operation samt den dafür notwendigen Parametern.

Listing 8-1
Code zum Ausführen eines CSRF-Angriffs

```
...
<img
  src="http://www.site.com/admin/changeLimit?id=5&limit=10000"
  width="0" height="0">
...
```

CSRF benötigt eine aktive Session.

Besucht nun der Administrator der Webanwendung diese Seite – ob im selben Browserfenster oder einem Tab spielt keine Rolle – kann dieser Request erfolgreich ausgeführt werden. Schließlich ist der Administrator bereits bei der anzugreifenden Webanwendung auf `www.site.com` angemeldet. Anstatt des nicht vorhandenen Bildes wird maximal der Platzhalter für ein nicht gefundenes Bild angezeigt; meist verhindert die auf `0` gesetzte Breite und Höhe des `img`-Tags aber jegliche Anzeige. Die angegriffene Webapplikation erhält damit die Aufforderung, das Limit des Benutzers `Marvin` auf `10000` zu erhöhen. Eine dazu passende Statusänderung für diesen Benutzer in die Gruppe `A` würde einen weiteren Request mit einem ähnlichen Link erfordern. Der Administrator hat so unwissentlich und unwillentlich einen Benutzer verändert. Im Log (sofern eines existiert, das solche Änderungen aufzeichnet) würde nur der Administrator als Bearbeiter auftauchen und nichts auf einen Angriff hindeuten.

Besucht ein anderer Benutzer ohne Konto, mit inaktiver Session oder mit unzureichenden Rechten bei der anzugreifenden Webanwendung dieselbe Seite, passiert einfach nichts. Aufseiten der angegriffenen Webapplikation kommt ein Request an, den die Anwendung mangels gültiger Session oder unzureichenden Rechten einfach verwirft und eventuell protokolliert.

Webanwendungen mit lange gültigen Sessions sind interessant.

Grundsätzlich für CSRF interessant sind Webanwendungen mit einer ausreichend langen Sessiondauer, die gleichzeitig bei den auszuführenden Aktionen keine weiteren Zugangsdaten abfragen, z. B.

nochmals das Passwort oder sogar eine individuelle TAN. Diese *Re-Authentication* genannte Maßnahme würde einen CSRF-Angriff ansonsten aufhalten. Im Idealfall für den CSRF-Angreifer meldet sich der Administrator der Webanwendung oder ein anderer Benutzer mit ausreichend Rechten morgens als Erstes bei einer solchen Webanwendung an und bleibt den ganzen Tag über angemeldet. Der Angreifer besitzt zwingend (umfangreiche) Kenntnisse über diese Applikation, vor allem aber kennt er deren Formulare und die verschiedenen direkt aufrufbaren Operationen samt deren notwendigen URL-Parametern. Im Webumfeld sind das beispielsweise prinzipiell für jeden verwendbare Webmailer oder Onlineshops. In Intranets sind es sehr viele standardisierte und damit weitverbreitete Unternehmensanwendungen. Individuelle Webanwendungen sind ebenso Ziel von CSRF-Angriffen, allerdings hat hier der Angreifer meist einen deutlich höheren Aufwand, um Informationen über Formulare, Links samt deren Parametern und kritische Operationen herauszufinden. Einen ausreichenden Schutz vor CSRF stellt eine öffentlich weitgehend unbekannte Individualsoftware aber dennoch nicht dar.

CSRF kann Anwendungen im Intranet erreichen.

Gelegentlich ist die ausgewählte Webanwendung überhaupt nicht über das Internet erreichbar, ihre Benutzung also beispielsweise auf das unternehmenseigene Intranet beschränkt. Ein Angreifer hat somit keine direkte Zugriffs- und Angriffsmöglichkeit auf die ausgesuchte Webanwendung. In diesem Fall ist es für ihn natürlich deutlich schwieriger, Detailinformationen über verwundbare Operationen herauszufinden. Ex-Mitarbeiter oder ein am Projekt beteiligter (externer) Entwickler, Social Engineering und vor allem der Angriff auf Standardwebanwendungen sind hierfür nutzbare Informationsquellen.

8.2 Was kann passieren?

Ein CSRF-Angriff kann unterschiedliche Folgen haben, die von den Möglichkeiten der angegriffenen Webanwendung und den Rechten des dabei unrechtmäßig verwendeten Benutzers abhängen. Sofern ein legitimer Benutzer keine Rechte zur Durchführung einer Aktion besitzt, kann ein Angreifer per CSRF ebenfalls nichts anrichten. Hier kann er nur weiter auf einen Benutzer mit ausreichenden Rechten warten.

Bei CSRF geht es wie gesagt darum, Aktionen im Namen eines Benutzers auszulösen. Das kann eine Bestellung sein, das Veröffentlichen von Nachrichten auf der Unternehmenswebseite oder das Manipulieren bestimmter Datensätze.[3] Bei der Manipulation von Daten ist prinzipiell

[3] Oder beispielsweise das heimliche Aktivieren der Webcam samt Upload des Videos in eine Online-Plattform: http://heise.de/-1776339

alles denkbar: vom Anlegen über das Bearbeiten bis hin zum Löschen – also beispielsweise auch das Ändern eines Benutzerpassworts, sofern die Webanwendung bei dieser Operation nicht die Eingabe des aktuellen Passworts erfordert. Damit hat der Angreifer vollständigen Zugriff auf das Konto des rechtmäßigen Besitzers und hat diesen gleichzeitig ausgesperrt.

CSRF erlaubt nicht das Auslesen von Daten.

Das Lesen von beliebigen Daten wie bei der SQL Injection (Abschnitt 6.2) ist dagegen nicht möglich.[4] Der Grund dafür ist, dass die gelesenen Daten ja zum Angreifer zurückgeliefert werden müssten, aber der unschuldige Benutzer den Request ausgelöst hat. Sofern der Server eine verwertbare und anzeigbare Antwort liefert, geht diese immer nur zum angemeldeten Benutzer zurück und würde diesem wahrscheinlich zumindest merkwürdig vorkommen. Ein Angreifer erhält somit keinen Zugriff auf die von der Webanwendung zurückgeschickten Daten.

Für den Fall, dass durchgeführte Änderungen in der Webanwendung protokolliert werden, deuten diese immer auf den angemeldeten Benutzer hin und nicht auf den Angreifer. Der dabei verwendete Benutzer wird nun häufig fälschlicherweise verdächtigt, diese Aktionen selbst ausgeführt zu haben. Der Beweis des Gegenteils gestaltet sich entsprechend schwierig. Auf dem Computer des Benutzers ist keine Schadsoftware vorhanden, es wurden weder Viren noch Trojaner installiert. Der Benutzeraccount selbst wurde nicht kompromittiert, und im Anwendungslog taucht nur dieser Benutzername zu normalen Arbeitszeiten auf. An die Untersuchung der Webapplikation in Hinblick auf eine Cross-Site-Request-Forgery-Verwundbarkeit wird meist erst gedacht, wenn mehrere Benutzer vom selben Problem betroffen sind oder ein erfahrener Benutzer oder Entwickler diese Sicherheitslücke entdeckt und meldet.

Angriff auf Hardware

Im Zuge der immer weiter um sich greifenden Heimautomatisierung (beispielsweise komplett steuerbar in einer Webanwendung vom eigenen Rechner oder Mobilgerät) ist es genauso denkbar, dass zukünftig auf Hardware in Form von Heizungen, Strom oder anderen elektronischen Geräten per CSRF-Angriff zugegriffen wird. Stellen Sie sich vor, nach einem Klick auf einen E-Mail-Link wird Ihre Zentralheizung im Sommer auf maximale Stärke gestellt oder das System geht im tiefsten Winter in den Reparaturmodus und schaltet sich vollständig ab. Selbst »individuelle« Webanwendungen einzelner Benutzer können sich so zu lohnenswerten Zielen entwickeln.

[4] Allerdings lässt sich natürlich auch ein SQL-Injection-Angriff per CSRF durchführen und als URL-Parameter Teile eines SQL-Statements injizieren. Das Ziel ist dabei nicht das Auslesen, sondern das Manipulieren von Daten.

8.3 Wie läuft ein Angriff ab?

Das Hinterhältige an CSRF-Angriffen ist, dass der Angreifer nicht selbst die Webanwendung angreift, sondern einen legitimen Benutzer dazu verwendet. Sein einziger eigener Kontakt mit der Webanwendung ist das Ausspionieren der Formulare bzw. Links, genauer gesagt das Ausspionieren von deren Aufbau (Parameter) und den dahinterliegenden URLs – zumindest solange ihm die Webanwendung unbekannt ist. Der Angreifer benötigt in jedem Fall ausreichende Kenntnisse über die anzugreifende Webanwendung. Bei Standardanwendungen ist der Angriff somit deutlich einfacher durchzuführen. Das Auswählen von geeigneten Operationen und das Testen der URLs kann in diesem Fall gegen eine (Test-)Webanwendung erfolgen und später auf die anzugreifende Webanwendung übertragen werden.

Das weitere Vorgehen folgt anschließend immer einem mehr oder weniger identischen Muster. Den grundsätzlichen Ablauf eines CSRF-Angriffs zeigt Abbildung 8-2.

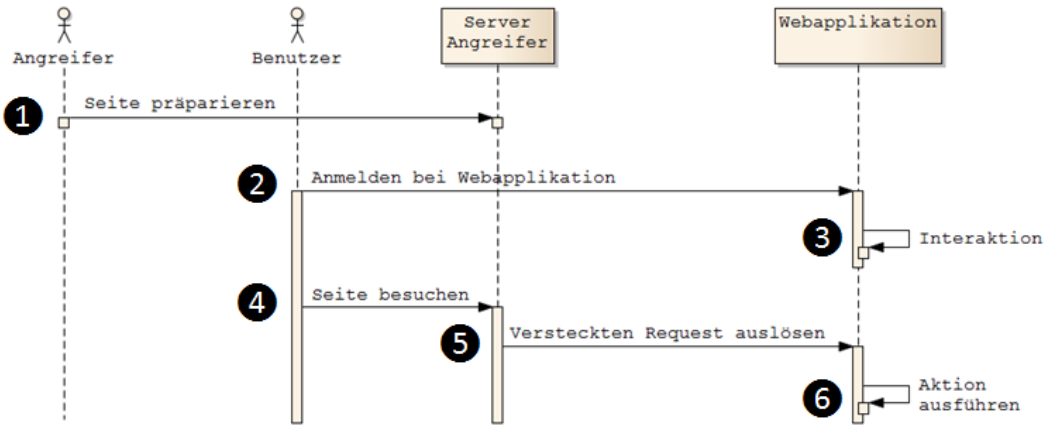

Abb. 8-2
Ablauf eines CSRF-Angriffs

Nachdem der Angreifer sich über die anzugreifende Webanwendung informiert hat, präpariert er beispielsweise eine Nachricht in einem Forum oder eine unverdächtige Webseite und veröffentlicht diese (1). Seltener schickt er eine E-Mail mit einem manipulierten Link an den oder die Benutzer. In vielen Fällen wird der CSRF-Angriffscode dabei einfach anstelle einer echten Bildressource in ein `img`-Tag eingebettet. Möglich sind ferner die Verwendung von `script`-Tags oder das Verstecken von Formularen in einem `Frame` oder `iFrame` mittels Cascading Style Sheets inklusive deren Versand per JavaScript. Im Gegensatz zum `img`-Tag werden Skripte und Frames aber häufiger vom Browser oder einem Add-on blockiert und nicht automatisch ausgeführt oder können aufgrund von

Browsereinstellungen nicht verwendet werden. Das `img`-Tag verspricht daher beim Angriff auf simple HTTP-GET-Requests häufiger zum Erfolg zu führen. Doch egal welches Tag der Angreifer letztendlich verwendet, nachdem der Köder ausgelegt ist, wartet er zunächst einmal auf seine Opfer.

Die Benutzer Ihrer Webanwendung melden sich in der Zwischenzeit dort an (2) und arbeiten normal mit der Anwendung (3). Irgendwann besucht einer der Benutzer nebenbei die vom Angreifer vorbereitete Seite (4). Dieser Aufruf löst automatisch einen Request an die anzugreifende Webanwendung aus (5), beispielsweise über ein Bild, das von diesem Server geladen werden soll. Eingebettet in die Bild-URL ist dabei der auszuführende Request samt den notwendigen Parametern, wie einer ID oder einem Namen. Der Browser des Opfers schickt beim vermeintlichen Laden des Bilds automatisch alle zu dieser Domain gehörenden Cookies samt der darin gespeicherten Session-ID des Benutzers (den gespeicherten Zugangsdaten) mit zur Webanwendung. Die angegriffene Webanwendung verifiziert die Session-Informationen des korrekt angemeldeten Benutzers und führt danach die im Request angegebene Operation aus (6) – ganz so, als ob der Benutzer selbst auf den entsprechenden Link geklickt oder das Formular ausgefüllt und abgeschickt hätte.

Der Benutzer bekommt von alldem nichts mit, denn alle Operationen werden heimlich im Hintergrund ausgeführt. Sein Webbrowser zeigt weiterhin die zuletzt aufgerufene Webseite des Angreifers an.

8.4 Was können Sie dagegen tun?

Sofern Sie das Buch von Anfang an gelesen haben, erwarten Sie jetzt vielleicht, dass ich Sie wieder einmal auf die Notwendigkeit der Input-Validierung (Abschnitt 4.2) hinweise. So hilfreich und wichtig die Validierung an vielen Stellen auch ist, bei Cross-Site Request Forgery nützt sie leider ebenso wenig wie das Output-Escaping (Abschnitt 4.3). Die Ursache dafür ist, dass Sie bei CSRF nicht mit ungültigen oder irgendwie gefährlichen Eingabedaten zu kämpfen haben, sondern mit ungewollten Requests, die gültige Daten beinhalten. Prinzipiell müssen Sie in Ihrer Webanwendung nur dafür Sorge tragen, dass sie zwischen gewollten Benutzerrequests und ungewollten (heimlich untergeschobenen) Angreiferrequests unterscheiden kann.

Gleichzeitig handelt es sich bei CSRF-Angriffen nicht um das Ausnutzen typischer Programmierfehler wie beispielsweise bei der SQL Injection oder beim Cross-Site Scripting. Der Entwickler hat hier keinen Bug im eigentlichen Sinne in die Software eingebaut, sondern hat nur