Ansatz zur Erkennung von HTTPS Stripping Attacken

Approach to Recognize HTTPS Stripping Attacks
Bachelor-Thesis von Tino Fuhrmann
10.02.2012

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Sicherheit in der Informationstechnik

CASED

Ansatz zur Erkennung von HTTPS Stripping Attacken
Approach to Recognize HTTPS Stripping Attacks

Vorgelegte Bachelor-Thesis von Tino Fuhrmann

Prüfer: Prof. Dr. Waidner
Betreuer: Marco Ghiglieri

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 10. Februar 2012

_____

(Tino Fuhrmann)

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Wer kennt es nicht, dass man Bankgeschäfte erledigen musste und vor verschlossenen Türen der Filiale stand. Durch die globale Verbreitung des Internets nutzte die Bank ihre Chance dem Kunden die Möglichkeit zu bieten außerhalb der Geschäftszeiten Geldgeschäfte abzuwickeln. Das Online-Banking war geboren.

Die Banken warben mit: „OnlineBanking einfach und sicher für Jedermann“

Um Geldgeschäfte zu tätigen muss eine sichere Verbindung zwischen dem Benutzer und der Bank bestehen. Dazu wurde das Secure Sockets Layer, kurz (SSL), Protokoll entwickelt. Die Sicherheit der Verbindung zwischen zwei Teilnehmern wird durch eine Verschlüsselung garantiert.
Da die Kriminalität vor dem Internet kein Halt macht, war es eine Frage der Zeit, bis die als sicher geglaubte Verbindung zur Bank missbraucht werden würde. In dieser Bachelorabeit sollen zwei Möglichkeiten des Missbrauches aufgezeigt werden. Die Gemeinsamkeit beider Varianten besteht darin, dass ein Dritter sich in die Verbindung einschleust. In der einen Variante wird bei einer Anwahl einer sicheren Verbindung eine unsichere Verbindung aufgebaut. Bei der Anderen wird eine sichere Verbindung zum Dritten anstatt zur Bank aufgebaut.
Um den Benutzer und die Bank vor solchen Angriffen und dem daraus resultierenden Schaden zu schützen, ist die Idee entstanden eine Sicherheitslösung zu entwickeln. In Zusammenarbeit mit dem Fachbereich *Sicherheit in der Informationstechnik* ist die Bachelorarbeit **Ansatz zur Erkennung von HTTPS Stripping Attacken** entstanden.

1.1 Problemstellung und Zielsetzung

Die Problemstellung in dieser Arbeit besteht darin, eine Methode zu entwickeln, die erkennen soll, ob das Hyper Transfer Protocol Secure (HTTPS) verwendet wird und das Originalzertifikat noch vorhanden ist.
Dazu wird eine Firefoxerweiterung und ein PHP Code für den Webserver entwickelt. Dem Benutzer soll optisch angezeigt werden, wenn er mit einer unsicheren Verbindung surft. Falls sich ein Dritter in die Verbindung einklinkt, wird dies erkannt und dem Benutzer angezeigt. Des Weiteren werden die versendeten Daten unabhängig von der Verbindung verschlüsselt, sodass nur der Webserver die Möglichkeit besitzt die Daten zu entschlüsseln.

1.2 Aufbau der Arbeit

Die Arbeit ist in einem schriftlichen und praktischen Teil aufgeteilt.
Der schriftliche Teil ist in fünf Kapitel angeordnet, die wie folgt gegliedert sind. Das erste Kapitel stellt eine kurze Einführung in die Thematik dar. Das zweite Kapitel befasst sich mit dem grundlegenden Begrifflichkeiten. Beginnend was ein Webbrowser ist, bis hin was Hypertext Transfer Protocol (HTTP) bzw. Hypertext Transfer Protocol Secure (HTTPS) sind. Es werden auch aktuelle Angriffe auf das Secure Sockets Layer (SSL) Protokoll vorgestellt. Im dritten Kapitel werden die zwei verschiedenen HTTPS Stripping Attacken präsentiert. Im fünften Kapitel wird der praktisch, entwickelte Teil vorgestellt. Des Weiteren werden die Angriffe im davor liegenden Kapitel angewandt, durchgeführt und analysiert. Im letzten Kapitel wird ein Fazit, sowie ein Ausblick, wie der praktische Teil weiter zu verbessern ist, vorgestellt. Am Ende dieser Arbeit ist der Code der praktischen Arbeit einzusehen.

2 Grundlagen

In diesem Kapitel werden die Grundlagen für eine sichere Übertragung von Webseiten erläutert. Ein Server in dieser Arbeit ist immer ein Web-Server mit spezieller Server-Software, die Hypertext Markup Language (HTML) Seiten ausliefert. Auf der Clientseite wird eine HTML Seite gewöhnlich von einem Browser interpretiert und für den Benutzer aufbereitet. Zuerst folgt eine kurze Einleitung zum World Wide Web. Anschließend ist das Kapitel in Server- und Clientseite aufgeteilt. Zuletzt werden Grundlagen für die HTTPS Stripping Attacken erläutert.

2.1 World Wide Web

Das World Wide Web (WWW) integriert bestehende Internetdienste durch eine einheitliche Adressierung und Bedienung sowie durch ein standardisiertes Dokumentenformat. Man unterscheidet WWW-Objekte, -Server und -Browser. WWW-Objekte können Dokumente oder beliebige Datenbestände wie Nachrichten und Datenbanken sein. Spezielle WWW-Objekte sind die Web-Seiten, die in HTML geschrieben und mittels des Hypertext Transport Protokolls (HTTP) übertragen werden. Web-Seiten sind Multimediadokumente, die insbesondere Verweise (engl. *link*) auf andere WWW-Objekte enthalten können. Durch diese Verweise entsteht ein weltweites Geflecht von Objekten, woraus der Name Web (deutsch: Netz) abgeleitet wurde.[6]

2.2 Serverseite

2.2.1 Webserver Software

Die Aufgabe einer Webserver Software besteht darin, die Anfragen eines Clients zu verarbeiten und ihm nach der Bearbeitung eine Antwort zurückzuschicken.
Auf Webservern wird die Webserver Software installiert, um deren Aufgaben ausführen zu können. Diese Server sind mit dem Internet verbunden. Sie stellen unter anderem Webseiten, Dokumente und Objekte bereit. Diese Informationen können mithilfe eines Webbrowsers vom Client angefordert werden.
Jeder Server besitzt eine eindeutige IP-Adresse und eventuell einen Domainnamen im Internet. Die Kommunikation erfolgt durch die Protokolle HTTP oder HTTPS, was in den nachfolgenden Kapiteln erklärt wird.

2.2.2 HTTP

Das Hypertext Transfer Protocol [3], kurz HTTP, ist ein zustandsloses Protokoll in der Anwendungsschicht. Seit den 90er Jahren wird es im World Wide Web benutzt. Es ist ein sehr einfach gehaltenes Protokoll, was den Transfer von Rohdaten über das Internet erlaubt. Webbrowser verwenden es, um auf Webserver zuzugreifen. HTTP ermöglicht das Interagieren mit Webservern. Die Darstellungsform sieht wie folgt aus *http://www.tu-darmstadt.de*. Alle Clients und Server müssen die Richtlinien des Protokolles einhalten.

Methode	Beschreibung
GET	Anforderung zum Lesen einer Webseite
HEAD	Anforderung zum Lesen des Headers einer Webseite
PUT	Anforderung zum Speichern einer Webseite
POST	Anhängen an eine benannte Ressource (z.B eine Webseite)
DELETE	Entfernen einer Webseite
TRACE	Echo für die eingehende Anforderung
CONNECT	Für zukünftige Verwendung reserviert
OPTIONS	Abfrage bestimmter Optionen

Tabelle 1: In HTTP integrierte Anforderungsmethoden

2.2.3 HTTP Verbindung

Eine Kommunikation zwischen Client und Webserver erfolgt durch den Austausch von HTTP Nachrichten. Diese Nachrichten übermitteln die Anfragen (Request) und Antworten (Response) zwischen Client und Server. Client und Server bauen gewöhnlicherweise auf dem Standardport 80 eine Verbindung auf, da der Webserver standardmäßig eine unverschlüsselte HTTP Verbindung auf diesem Port erwartet. Bei HTTP Nachrichten unterscheidet man die in Tabelle 1 aufgelisteten Methoden. Für die weitere Ausarbeitung ist ausschließlich POST interessant.

2.2.4 POST

Die POST Methode dient dazu Formulardaten vom Client an den Server zu übermitteln. Die Daten werden als Teil einer HTTP Nachricht versendet.
Die Daten können wie folgt ausschauen: $name = Max\&passwort = 123$;
Dabei wird zuerst der Name der Variable genannt *name*, anschließend folgt nach dem Gleichheitszeichen der Wert, der vom Client im Formular eingegeben wurde, Max. Weitere übermittelte Werte werden mittels einem & getrennt. [29]

2.2.5 SSL/TLS

Das Secure Socket Layer Protokoll, kurz SSL, wurde Mitte der 90er Jahren von Netscape Communications Corporation entwickelt. Die aktuellste Version von SSL ist 3.0. Seit der Version 3.1 wird das SSL Protokoll unter dem neuen Namen Transport Layer Security[1], kurz Transport Layer Security (TLS), weiterentwickelt und standardisiert. In der weiteren Arbeit wird die Abkürzung SSL für beide Bezeichnungen verwendet. [22]
SSL ist ein hybrides Verschlüsselungsprotokoll zur Datenübertragung im Internet. Eine hybride Verschlüsselung bezeichnet dabei die Kombination von symmetrischen und asymmetrischen Kryptoverfahren.
Das Ziel von SSL ist es Vertraulichkeit zwischen dem Benutzer und dem Webserver herzustellen. Dabei liegen die Hauptaufgaben des Protokolls bei [6]:

[1] TLS 1.0 steht neu für SSL 3.1

1. Authentifizierung zwischen Client und Server
2. Entscheidung welche Parameter Client und Server unterstützen
3. Verschlüsselung der Kommunikation
4. Überprüfung der Datenintegrität

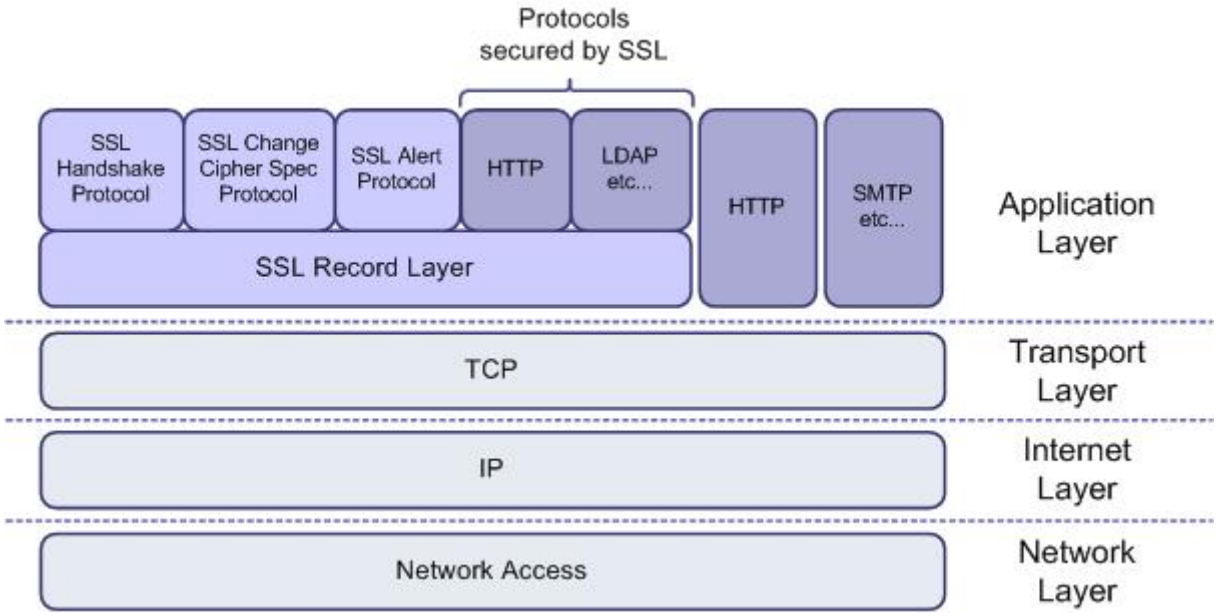


Abbildung 1: SSL in einem Auszug des OSI Schichtenmodell [14]

SSL befindet sich im OSI Schichtenmodell (Abb. 1) über der Transportschicht (TCP) und ist unterhalb der höheren Protokolle wie HTTP einzuordnen. Dieses Design hat den Vorteil, dass SSL transparent und unabhängig von den verwendeten Applikationsprotokollen eine abgesicherte Verbindung herstellen kann. [17]
Der allgemeine Ablauf bei Verbindung eines Clients mit einer per SSL gesicherten Webseite ist folgender: Der Webserver sendet im ersten Schritt sein digitales Zertifikat. Sofern dieses von einer Certificate Authority (CA) ausgestellt und signiert wurde und deren digitales Zertifikat schon im Browser des Clients gespeichert ist, erkennt dieser es als gültig an. Die Standardbrowser der Firmen Microsoft und Mozilla bringen schon von Hause aus einige Zertifikate von bekannten, großen CAs mit. Für den Fall, dass das Zertifikat der zertifizierenden CA dieser Webseite nicht enthalten ist, bleibt dem Benutzer die Entscheidung überlassen, dem Webserverzertifikat zu vertrauen. Dabei besteht natürlich die Gefahr, dass Benutzer jederzeit auch ungültige Zertifikate annehmen können und diesen dann in Zukunft fälschlicherweise Vertrauen schenken. Die Sicherheit hängt in solchen Fällen alleine von den Benutzern selbst ab. Unternehmen, welche aus wirtschaftlichen Interessen oder aus Gründen des Vertrauens eigene CAs betreiben, anstatt ihre Server von einer der großen, bekannten CAs zertifizieren zu lassen, sind deshalb darauf angewiesen, dass alle Clientrechner vor der ersten Verbindungsaufnahme mit dem Server, das Zertifikat ihrer eigenen CA in die Liste vertrauenswürdiger CAs importieren. [19]
Das SSL Protokoll umfasst zwei Teilprotokolle: das Handshake Protokoll hat die Aufgabe eine sichere Verbindung aufzubauen, das Record Protokoll verwendet anschließend die aufgebaute Verbindung.

2.2.6 SSL Handshake Protokoll

Das Handshake Protokoll kann in vier Phasen aufgeteilt werden. In der darauffolgenden Abbildung 2 soll das Handshake Protokoll anschaulich dargestellt werden.

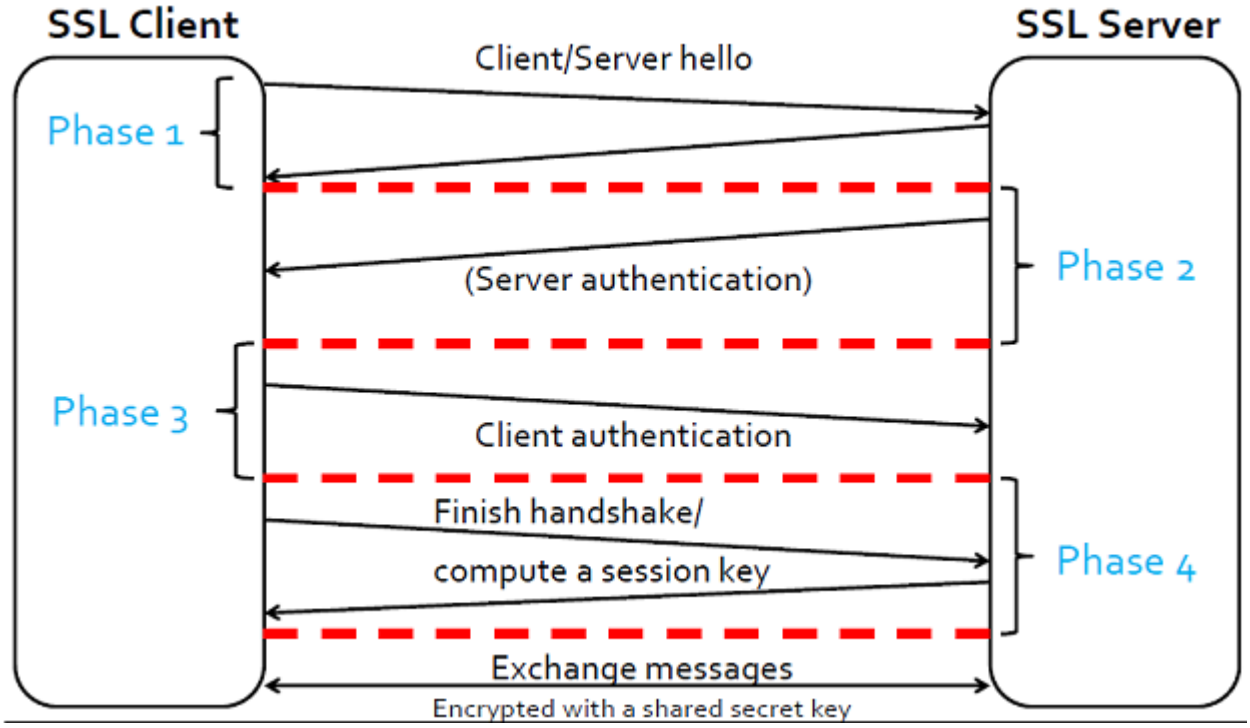


Abbildung 2: SSL Handshake Protkoll mit seinen vier Phasen [26]

In Phase 1 kommt es zu einer Verbindungsanforderung vom Client zum Server. Zusätzlich werden die Präferenzen bezüglich der Komprimierung, der kryptografischen Algorithmen und der verwendeten SSL Protokollversion ausgehandelt. Der Server wählt hierzu die sichersten gemeinsamen Methoden aus. Anschließend in Phase 2, der Serverauthentifikation, sendet der Server sein Zertifikat mit dem öffentlichen Schlüssel zum Client. Das Zertifikat dient dazu, sich gegenüber dem Client zu authentifizieren. Nur wenn der Client das Zertifikat akzeptiert, kommt die Verbindung zu Stande. Nun folgt Phase 3, die Clientauthentifikation. Hierbei bestätigt nur der Client, dass er dem Zertifikat vertraut und dass die ausgehandelten Präferenzen akzeptabel sind. Die vierte Phase ist dafür gedacht, dass Handshake Protokoll abzuschließen und zum Datenaustausch zu kommen. Es wird ein Sitzungsschlüssel („session key“) generiert, der während der Verbindung zum Ver- und Entschlüsseln der Daten genutzt wird. Alle Daten werden von nun an verschlüsselt übertragen.[6, 26]

2.2.7 Public-Key Verfahren

Das Public-Key Verfahren ist ein asymmetrisches Verschlüsselungsverfahren, durch dass Nachrichten signiert und verschlüsselt werden können. Das Signieren garantiert, dass die Nachricht auch wirklich vom angegeben Absender stammt. Zum Signieren und Verschlüsseln werden zwei Schlüssel benötigt. Es wird ein Schlüsselpaar mit einem öffentlichen Schlüssel (public Key) und einem privaten Schlüssel (private Key) generiert. Der öffentliche Schlüssel wird zur Verschlüsselung verwendet und muss nicht geheim gehalten werden. Er kann an beliebig viele Personen weitergegeben

oder in einem Verzeichnis öffentlich zugänglich gemacht werden. Der private Schlüssel dient der Entschlüsselung, er muss daher privat, d.h. geheim gehalten werden.
Der Vorteil liegt darin, dass es praktisch unmöglich ist, aus dem öffentlichen Schlüssel den privaten Schlüssel zu ermitteln. Der private Schlüssel bleibt nur dem Inhaber des erzeugten Schlüsselpaares bekannt. Um die Verbreitung öffentlicher Schlüssel auf sichere und zuverlässige Weise zu ermöglichen, wurden digitale Zertifikate geschaffen. [19]

2.2.8 Public-Key Infrastruktur

Die Aufgabe der Public-Key Infrastruktur (PKI) ist die Generierung und anschließende Zertifizierung von Schlüsselpaaren sowie die Verteilung von öffentlichen Schlüsseln. Mithilfe der PKI wird Authentizität, Vertraulichkeit und Integrität gewährleistet. Authentizität garantiert, dass eine Person die ist, für die sie sich ausgibt. Dies wird mittels einer vertrauenswürdigen CA beglaubigt.
Vertraulichkeit garantiert, dass die übertragenen Daten nur autorisierte Personen lesen können.
Die Integrität ist dafür verantwortlich, dass vertrauliche Daten bei der Übertragung nicht veränderbar sind. [19]

2.2.9 Zertifikate

Ein digitales Zertifikat ist eine signierte Datenstruktur, die neben einem öffentlichen Schlüssel noch den Namen seines Besitzers enthält. Dadurch wird eine sichere Zuordnung des Schlüssels zu einem Webserver et cetera ermöglicht. Zertifikate selbst sind weder geheim noch geschützt. Das Signieren eines Zertifikates wird anhand von verschiedener vertrauenswürdiger Einrichtungen erreicht. Dabei muss eine solche Einrichtung das Vertrauen aller an der Kommunikation beteiligten Personen aufweisen können, inklusive dem zertifizierten Selbst. Auf Basis dieser Vertrauensbeziehung wird gewährleistet, dass der signierte öffentliche Schlüssel auch wirklich zur zertifizierten Person oder Webserver gehört und für die sichere Kommunikation mit diesem verwendet werden kann. Wird andernfalls dem Zertifikat kein Vertrauen geschenkt, kommt es zu keiner Verbindung beziehungsweise Schlüsselaustausch.
In unserem Fall werden Zertifikate basierend auf dem Standard X.509 Version 3, sogenannte Extended Validation SSL Zertifikate (EV-SSL), verwendet. Diese Zertifikate sind an strenge Vergabekriterien gebunden. Anhand der EV-SSL Zertifikate soll auf dem ersten Blick dem Benutzer gezeigt werden, dass es sich um eine sichere Verbindung handelt. Ein EV-SSL Zertifikat beinhaltet folgende Informationen: [19]

- Den Namen des Servers, für den ein Zertifikat erstellt wurde
- Den Namen der Organisation, die das Zertifikat ausgestellt hat
- Die Seriennummer des Zertifikates
- Die Angaben zum verwendeten kryptographischen Algorithmus, mit dem das Zertifikat von einer Zertifizierungsstelle signiert wurde
- Das Gültigkeitszeitraum; mit einem Anfangs- und Enddatum

- Den öffentlichen Schlüssel des Servers sowie den Namen des Algorithmus dieses Schlüssels
- Die Liste der Verwendungszwecke, für die das Zertifikat ausgestellt wurde

2.2.10 HTTPS

Das Hyper Text Transfer Protocol Secure, kurz HTTPS, ist ein sicheres Protokoll zur Übertragung von Daten im Internet. Es wird zur Verschlüsselung und zur Authentifizierung der Kommunikation zwischen Webserver und Browser im Internet verwendet. Das Ziel hierbei ist es, dass kein Subjekt[2] die versendeten Daten mitlesen kann, was vor allem das Online-Banking sicherer gegenüber Fälschungen macht.
Im Prinzip ist HTTPS eine Zusammensetzung der Protokolle HTTP und SSL. Es werden standardmäßig X.509-Zertifikate, sogenannte Extended Validation SSL Zertifikate [20], eingesetzt. Des Weiteren sollte erwähnt werden, dass HTTPS Seiten nicht direkt für Sicherheit stehen, da sie nur die Daten verschlüsselt übertragen. Es wird damit nur das Auslesen der Daten für ein drittes Subjekt erschwert, aber ist das Zertifikat unsicher oder vertraut man einem nicht vertrauenswürdigen Zertifikat, schützt HTTPS nicht davor, dass die Daten ausspioniert oder missbraucht werden. [19, 21]
Im folgenden werden die Darstellungsformen in den gängigsten Browsern betrachtet, wie eine HTTPS Verbindung in diesen jeweils ersichtlich abgebildet wird. Die hierbei behandelten Browser sind einmal Firefox, Internet Explorer und Chrome in ihrer aktuellsten Version. Die folgende Arbeit beschäftigt sich ausschließlich mit dem Firefox Browser.
Beim Firefox Browser wird eine HTTPS Verbindung farblich von der Adresszeile abgestuft. Deren

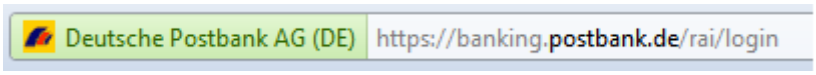


Abbildung 3: Firefox 9: Extended SSL Validierung

Abbildung 4: Firefox 9: Sichere Verbindungen

Ziel ist es kenntlich zu machen, für wen das Zertifikat ausgestellt wurde. Hierzu werden die Farben Grün und Blau benutzt, wobei durch Grün (Abb. 3) kenntlich gemacht wird, dass es sich um ein EV-SSL Zertifikat handelt. Durch die Farbe Blau (Abb. 4) wird gekennzeichnet, dass es sich um eine sichere Verbindung handelt.
Beim Chrome Browser werden Webseiten, die ein EV-SSL Zertifikat (Abb. 5) besitzen, farblich

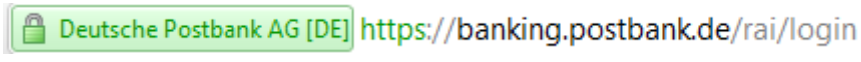


Abbildung 5: Chrome 16: Extended SSL Validierung

untermalt. Handelt es sich hingegen um kein EV-SSL Zertifikat, wird dies mit einem Schloss vor der Adresszeile symbolisiert (Abb. 6).
Beim Internet Explorer färbt sich bei einem EV-SSL Zertifikat die gesamte Adresszeile grün und

[2] Subjekt: in diesem Fall für eine Person, Server etc. stehen

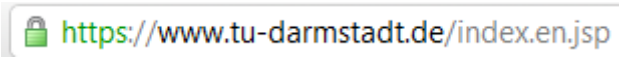


Abbildung 6: Chrome 16: Sicherheitsschloss Symbol bei gesicherten Verbindungen

Abbildung 7: Internet Explorer 9: Extended SSL Validierung

Abbildung 8: Internet Explorer 9: Sicherheitsschloss Symbol bei gesicherten Verbindungen

ein Schlosssymbol wird hinzugefügt (Abb. 7). Ist es kein EV-SSL Zertifikat, wird die Adresszeile nicht grün gefärbt, aber ein Schlosssymbol am Ende der Adresszeile angezeigt und damit symbolisiert, dass es sich um eine sichere HTTPS Verbindung handelt (Abb. 8).
Nachdem nun gezeigt wurde, wie eine HTTPS Verbindung in den verschiedenen Browsern dargestellt wird, wird kurz erläutert was für Vorbereitungen zum Aufbau einer HTTPS Verbindung getroffen werden müssen.
Damit eine HTTPS Verbindung aufgebaut werden kann, muss ein Zertifikat angenommen werden, welches die Echtheit des Webservers beweist. Nur mithilfe einer Zertifizierungsinstanz (CA) kann die Echtheit garantiert werden. Falls das Zertifikat nicht installiert sein sollte, erhält der Benutzer beim Öffnen der Webseite eine Fehlermeldung. Einige vertrauenswürdige CAs sind in jedem Browser vorinstalliert. In Firmen werden eigene Zertifikate noch in ihren Browsern installiert, um bestimmte Business Anwendungen fehlerfrei zu benutzen.
Zum Aufbau einer HTTPS Verbindung kommt es erst dann, wenn der Benutzer auf einen Link mit dem Merkmal *https:\\...* klickt oder die URL im Browser einträgt. Daraufhin wird eine Verbindung über den Port 443 zum Webserver aufgebaut. Der Webserver zeigt nun das Zertifikat an, woraufhin der Client anhand von den vorinstallierten CA Zertifikaten dieses Zertifikat auf Echtheit überprüfen kann. Anschließend erfolgt die nur für den Webserver lesbare Übertragung des temporären Sitzungsschlüssels. Mit dem auf beiden Seiten vorhandenen Sitzungsschlüssel kann eine Datenverschlüsselung starten. Es folgt eine kurze Aufzählung von Fehlern und Gefahren, die bei einer HTTPS Verbindung entstehen können. [21]

- Das Zertifikat ist abgelaufen
  Gefahr: Ein Angreifer nutzt das abgelaufene Zertifikat, dessen Privatschlüssel er erlangt hat.
- Das Zertifikat wurde von einer nicht vertrauenswürdigen Zertifizierungsinstanz ausgestellt.
  Gefahr: Ein Angreifer hat sich selbst ein Zertifikat ausgestellt.
- Der auf dem Zertifikat angegeben Servername stimmt nicht mit dem Webserver überein.
  Gefahr: Ein Angreifer besorgt sich ein für einen fremden Webserver ausgestelltes Zertifikat.

2.3 Clientseite

2.3.1 Webbrowser

Webbrowser wie der Microsoft Internet Explorer oder Firefox stellen Dienste zum Navigieren im WWW, zur Darstellung von Seiten sowie zur Interaktion mit Servern zur Verfügung. Mittels eines Browsers können WWW-Objekte von einem Server abgerufen, aber auch interaktive Benutzereingaben an den Server weitergeleitet werden. [6]
Eine Kommunikation zwischen einem lokalen Rechner (Client) und einem Webserver wird über das Internet bewerkstelligt. Mithilfe eines lokalen Programmes, dem Webbrowser, kann ein Verbindungsaufbau vom Client zum Webserver gestartet werden.

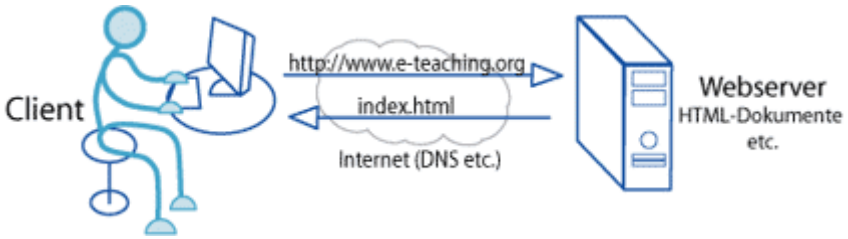


Abbildung 9: Kommunikation zwischen Client(-Browser) und Webserver [9]

Die Aufgaben eines Webbrowsers sind die Anfragen, die ein Benutzer an einen Webserver im Internet stellt, zu verarbeiten (Abb. 9). Dabei kann es sich bei den Anfragen um Webseiten, Datenbankzugriffe oder Dateien handeln. Des Weiteren ist der Webbrowser dafür da, Webseiten darzustellen und die empfangenen Daten zu verarbeiten.
Schaut man sich hierzu das verkürzte OSI Schichtenmodell (Abb. 1) an, ist der Webbrowser in der obersten Schicht des Schichtenmodelles anzusiedeln. Dies ist die Anwendungsschicht und sie stellt nur Funktionen und Protokolle für die Anwendungen zur Verfügung. In den darunterliegenden Schichten wird der eigentliche Datenübertragungs- und Verarbeitungsprozess abgewickelt.
Die Beziehung zwischen einem lokalen Rechner (Benutzer/Client) und einem Server wird als Client-Server-Infrastruktur definiert und beschreibt die Kommunikation der involvierten Rechnertypen. Dabei stellt der Client eine Anfrage zur Benutzung eines bestimmten Dienstes an den Server, der wiederum die Anfrage entgegennimmt, auswertet und den Client durch das Bereitstellen des gewünschten Dienstes bedient. [9]
Der Webbrowser ist für die Verarbeitung der empfangenen Daten und das Verschicken der Anfragen eines Benutzers zum Server zuständig. [20]
Die Kommunikation wird mithilfe des HTTP Protokolles durchgeführt.

2.3.2 Cookie

Das Cookie ist eine vom Webserver generierte Datenstruktur und beinhaltet Informationen über den zugreifenden Benutzer. Beim wiederholten Zugreifen auf die Webseite kann der Webserver den Cookie leicht auswerten. Der Webbrowser speichert die Cookies. Cookies werden vom Browser mit einer ID und einem Ablaufdatum versehen. Die Werte innerhalb eines Cookies haben immer die Form $[Name] = [Wert]$;. [6]

2.3.3 Javascript

JavaScript ist eine clientseitige Skript-Sprache. Das heißt, dass das Skript im Quellcode des Webdokuments eingebettet ist und erst vom Browser interpretiert und ausgeführt wird. Dies wird unabhängig vom Webserver durchgeführt. Durch Javascript besitzt man die Möglichkeit dynamischen Einfluss auf die Webseiten zu nehmen. Vorteilhaft für Javascript ist, dass die Sprache unabhängig von einer spezifischen Plattform ist. [10]

2.3.4 Erweiterungen für den Browser

Alle gängigen Browser haben die Möglichkeit eine Erweiterung zu implementieren. Der Vorteil liegt hierbei, dass die Funktionalität von den Browsern erweitert wird. Erst nach einer Installation sind diese im Browser aktiv.
In dieser Arbeit wird eine Erweiterung für den Firefox Browser erarbeitet. Diese werden durch XUL und JavaScript realisiert.
XUL steht für XMLUser-Interface Language. In Firefox wird XUL für die Definition der graphischen Benutzeroberfläche und für die Internationalisierung der Oberfläche verwendet. [7, 4]
Mittels JavaScript lassen sich XUL Elemente und HTML Elemente von der aktuellen Webseite verwenden. Durch die Erweiterungen haben sie vollen Zugriff auf den Inhalt der Webseite.

2.3.5 HTML

Die Hypertext Markup Language, kurz HTML, ist eine textbasierte Auszeichnungssprache zur Strukturierung von Inhalten wie Texten, Bildern und Hyperlinks in Dokumenten. HTML Dokumente sind die Grundlage des World Wide Web und werden von einem Webbrowser dargestellt. [28]

2.4 Angriffe

2.4.1 Man in the Middle Angriff

Der Man in the Middle Angriff, kurz MitM, ist ein Angriffsszenario, wo der Angreifer zwischen den Kommunikationspartnern sitzt und die Parteien glauben jeweils mit der entsprechenden anderen Partei ein Dialog zu führen, aber tatsächlich kommunizieren sie mit dem Angreifer. Das hat zur Folge, dass selbst verschlüsselte Verbindungen unverschlüsselt sind. Denn der Sender möchte mit dem Empfänger kommunizieren, doch statt die Daten vom Sender an den Empfänger zu senden, werden sie an den Angreifer versandt und dieser leitet sie weiter zum Empfänger. Schließlich denkt der Sender, dass der Angreifer der Empfänger ist und der Empfänger denkt der Angreifer ist der Sender. Nun kann der Angreifer alle Informationen abhören oder gar manipulieren.

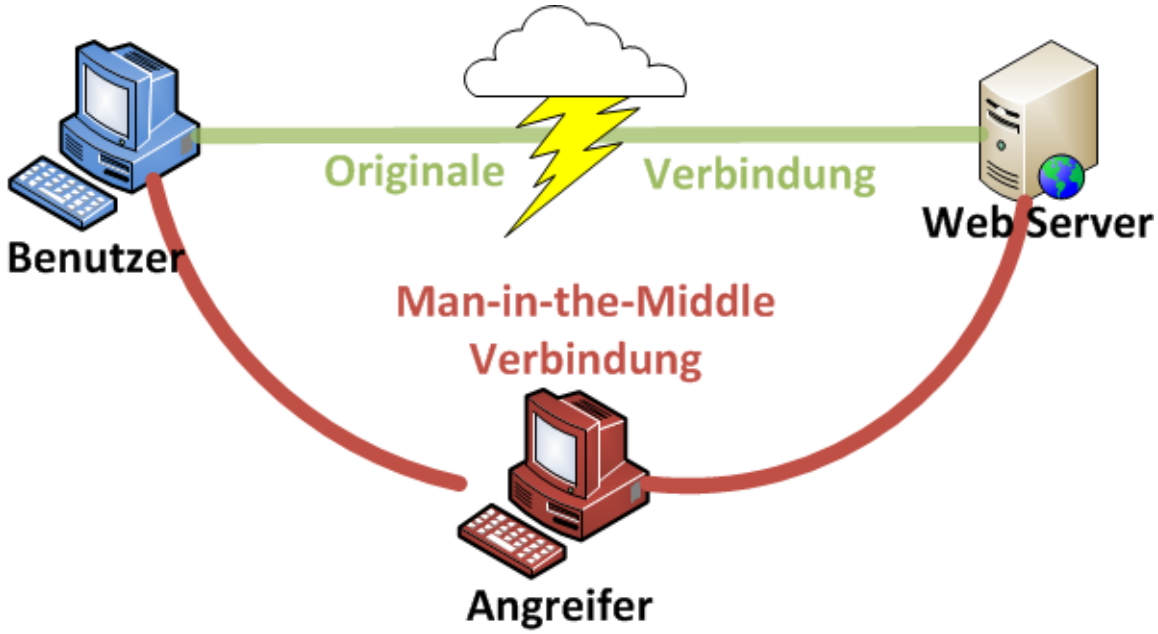


Abbildung 10: Man in the Middle Angriff

2.4.2 Homographischer Angriff

Unter einem homographischen Angriff versteht man, wenn die visuelle Ähnlichkeit zwischen zwei für die Identität ausschlaggebenden Namen, insbesonders bei Domainnamen, bewusst für Verwechselung genutzt werden. Für einen Benutzer ist es hierbei schwer, den Unterschied zwischen einem originalen Namen und einer Fälschung zu erkennen.
Mit der Einführung der internationalisierten Domainnamen (IDN) stehen außer dem uns bekannten ASCII-Zeichensatz eine Vielzahl von Schriften für Domainnamen zur Verfügung, die zum Teil eine Reihe ähnlicher Schriftzeichen enthalten. Damit vervielfachen sich die Möglichkeiten für homographische Angriffe.
Als Beispiel dazu empfiehlt es sich ähnliche Zeichen im ASCII Satz anzuschauen. Die Ziffer 0 und der Buchstabe O weisen Ähnlichkeiten auf und nur bei genauem Betrachten erkennt man den Unterschied. Des Weiteren sollte man das große I und das kleine l betrachten. Mit diesen Hin-

weisen könnte man einen visuell ähnlichen Domainnamen, wie z.b $IONEN$ $l0NEN$ erzeugen. [13]

2.4.3 Angriffe auf SSL

Trotz der vielversprechenden Sicherheit von SSL ist das Protokoll leider nicht von Angriffen verschont geblieben. Im Folgenden wird ein kurzer Einblick in verschiedene Angriffe auf SSL gegeben. Thai Duong und Juliano Rizzo [23] veranschaulichten bisher in ihrem Angriff, welche Informationen ein Angreifer im gleichen Netz über HTTPS übertragene Cookies ausspähen kann. Der Denkansatz beruht auf eine Schwachstelle, die seit dem Jahr 2001 bekannt ist und in allen SSL Versionen und in der TLS Version 1.0 noch vorhanden ist. Die Entwickler verwenden dazu eine Anwendung namens BEAST, welches aus zwei Komponenten besteht. Die erste Komponente ist entweder ein Javascript Code oder ein Java-Applets. Diese Komponente wird in den Browser des Opfers eingeschleust und ermöglicht präpierte Daten in die verschlüsselte Verbindung einzuschleusen. Bei der zweiten Komponente handelt es sich um einen Sniffer, der die übertragenen Daten belauscht. Der Angriff beruht auf einer Form des *block-wise-chosen-plaintext* Angriffes, welche dem Angreifer ermöglicht blockweise Klartexte seiner Wahl zu verschlüsseln. Aus den gewählten Klartexten und dem beobachteten Schlüsseltext lässt sich auf die verwendete Entropie schließen. Der Aufwand zum Brechen der Verschlüsselung verringert sich dadurch rapide.
Ein Update auf TLS 1.1 würde diesen Angriff nicht durchführbar machen, aber dennoch verwenden viele Server nur eine veraltete Version von TLS. Allerdings ist die Umstellung der Server offenbar nicht so einfach zu bewerkstelligen, da nicht alle Server und Browser die aktuellste Version unterstützen.
Eine weitere Angriffsmethode stammt von Marsh Ray und Steve Dispensa[16], die eine Methode zur Manipulation von SSL Verbindungen entwickelt haben. Die Angriffsmethode kann in Form eines *Man in the Middle* Angriffes ausgeführt werden. *MitM* Angriffe bedeuten, dass sich ein Angreifer bzw. eine dritte unerwünschte Partei zwischen einer Übertragung einklinkt. Selbst SSL Serverzertifikate schützen nicht vor einem solchen Angriff. Deshalb sollte es für jeden Benutzer wichtig sein, selbst einen Blick auf das Zertifikat seiner Homebanking-Seite zu werfen und zu kontrollieren, ob es das richtige Zertifikat ist. Dieser Angriff ist aber nicht so einfach durchzuführen. Ein Angreifer muss die Möglichkeit haben, sich bei dem Datenverkehr einzuklinken. Dies geht nur wenn er zum Beispiel beim Provider, Carrier oder einem Internetaustauschknoten sitzt. Dieses worst-case Szenario scheint unrealistisch zu sein, aber dennoch gibt es illoyale Mitarbeiter, die bei Providern und Carriern sitzen und einen Angriff durchführbar gestalten und somit ermöglichen Passwörter oder den Datenverkehr mitzuschneiden.
Der Angriff läuft wie folgt ab: Der Angreifer gibt sich als Server aus. Der ahnungslose Benutzer vertraut dem Server und gibt seine Daten ein. Mithilfe dieser Daten kann er bei Bezahlsystemen, die keine TAN Nummer besitzen, wie zum Beispiel Paypal, Geld transferieren. Dem Nutzer wird also eine sichere Verbindung und ein echte Webseite präsentiert. Zwar soll das Serverzertifikat helfen die Echtheit des Servers zu beweisen, aber die Angreifer sind so geschickt ihr gefälschtes Zertifikat von vertrauenswürdigen Zertifikatsaussteller wie Versign zu signieren, welche in den gängigsten Browsern vorinstalliert sind. Das führt dazu, dass selbst der Browser keine Warnung ausgibt und der Benutzer nicht gewarnt wird, dass es sich um einen nicht vertrauenswürdigen Server handelt oder gar nicht der Server ist mit dem man kommunizieren mag.
Aus den abgehandelten Angriffen wurde ersichtlich, dass SSL Verbindungen keine 100 prozentige Sicherheit gewährleisten. Obwohl sehr spezielle Voraussetzungen existieren müssen, dass solche Angriffe durchführbar sind, weist die Verbindung Schwachstellen auf. Ein blindes Vertrauen in die

Zertifikate hilft leider nicht. Der Benutzer trägt selbst die Verantwortung, ob er sich mit einer vertrauenswürdigen Webseite verbindet oder nicht oder ob er gar zu einer Webseite verbindet, die der vertrauenswürdigen Seite sehr ähnlich kommt. Der Benutzer muss hier selbst Initiative ergreifen, um festzustellen, ob die Webseite vertrauenswürdig ist oder nicht. Computerneulinge werden es schwer haben, eine vertrauenswürdige gegenüber einer nicht vertrauenswürdigen Webseite zu identifizieren.

3 HTTPS Stripping Attacken

Im folgenden Kapitel werden zwei verschiedene Ansätze von HTTPS Stripping Attacken erläutert. Beide Methoden beruhen auf einem *Man in the Middle* Angriff. Zuerst wird der Angriff SSLStrip[1] erklärt. Anschließend folgt der Angriff SSL Man in the Middle [5].

3.1 SSLStrip

SSLStrip ist eine Angriffsmethode mit der sich HTTPS Verbindungen im Browser abhören lassen. Die Methode basiert auf einer Kombination eines klassischen *Man in the Middle Angriff* (2.4.1) und einem homographischen Angriff (2.4.2). Zusätzlich wird auf die Unerfahrenheit des Benutzers im Umgang vom Erkennen von HTTPS Verbindung gesetzt.
SSLstrip selbst ist ein einfacher Proxy. Er baut darauf auf, dass der Benutzer nicht direkt auf HTTPS URLs aufruft, sondern mittels einer Portalseite wie *http://www.meine-bank.de* auf einen Link mit *https://homebanking.meine-bank.de* klickt. SSLStrip modifiziert jegliche HTTPS Links, so dass die SSL Verbindungen zu unverschlüsselten Verbindungen werden.

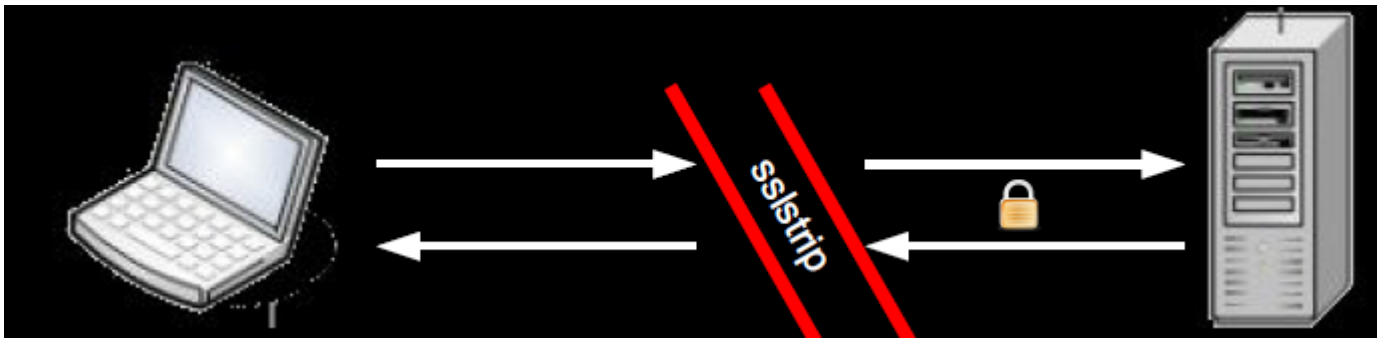


Abbildung 11: Ansatz des SSLStrip Angriffes von Black Hat DC 2009 [15]

Mit dem echten Server kommuniziert SSLStrip tatsächlich via HTTPS (Abb. 11). Nur die Kommunikation vom Benutzer zu SSLStrip wird über HTTP abgewickelt. Der Benutzer kann diesen Angriff daran erkennen, dass sein Browser anstatt über HTTPS sich über HTTP verständigt. Dennoch wird vom Benutzer der Browser oft nicht genau genug betrachtet. Wie erwähnt ist die Darstellung einer sicheren Verbindung bei Firefox mit einer farbigen Adresszeile gekennzeichnet. Zudem täuscht SSLStrip durch ein Favicon, dass die Form eines Schlosses (Abb. 12) hat die bekannten Sicherheitssymbole des Browsers, die bei einer HTTPS Verbindung der Browser in der Adresszeile dargestellt werden, vor. Zwar ist die Position des Schlosses falsch, dennoch wird dem Benutzer das Gefühl übermittelt, dass es sich um eine sichere Verbindung handelt.

Abbildung 12: Favicon von SSLStrip[15]

Dabei liegt die größte Schwierigkeit darin, einen Rechner zu überzeugen, dass er den SSLStrip Rechner als Proxy oder Router nutzen soll. Relativ einfach gehe das in lokalen Netzwerken. Denn hier könne man mit ARP-Spoofing arbeiten. So ließen sich in einem Firmennetz Zugangsdaten von Kollegen ausspionieren. Im Internet müsse man versuchen, den Nutzern Malware unterzuschieben,

die die Proxy Einstellungen des Browsers manipuliert. [1, 12]

3.2 SSL Man in the Middle Proxy

Der SSL Man in the Middle Proxy ist ein Java-basierter SSL Proxy, der als *Man in the Middle* (Abb. 13) agiert. Das bedeutet, er fängt HTTPS Requests ab und sendet sie dann an einen Webserver weiter. Der Proxy kommuniziert über eine HTTPS Verbindung mit dem Webserver. Zum Browser baut er eine weitere HTTPS Verbindung auf und benutzt ein selbst erzeugtes Zertifikat. Diese Zertifikate sind eine Kopie zu den vom Webserver bereitgestellten originalen Zertifikate. Sie unterscheiden sich nur bei ihrem Aussteller. Akzeptiert der Benutzer dieses gefälschte Zertifikat, besitzt der Proxy vollen Zugriff auf die Daten im Klartext.

Der Proxy hört vom Browser gesendete Connect Requests (Tab. 1) ab. Anschließend baut der

Abbildung 13: SSL Man in the Middle Proxy

Proxy eine HTTPS Verbindung zum Webserver auf und erhält von ihm das Zertifikat. Es wird ein gefälschtes Zertifikat erzeugt, wobei *SubjectDN, Serial Number, Extensions, etc.* übernommen werden. Er ändert ausschließlich nur *Aussteller, Public Key und Signatur*. Den privaten Schlüssel besitzt nun nur der Proxy und obwohl eine HTTPS Verbindung vorliegt, ist der Proxy in der Lage die übermittelten verschlüsselten Daten mitzulesen.

Zwar werfen die Browser eine Fehlermeldung, weil der Aussteller keiner von ihren vertrauenswürdigen Zertifikatsausteller beinhaltet, doch akzeptiert man es einmalig, kommt dieser Warnhinweis kein zweites Mal.

4 Szenario Lösung

In Kapitel 3 wurde gezeigt, welche Angriffe auf HTTPS existieren. Somit können diese Angriffe jederzeit durchgeführt werden. Da die meisten Angriffe das Online Banking betreffen, muss dieser sensible Bereich besonders geschützt werden. In Unternehmen ist es für einen Administrator jedoch ein leichtes Spiel eine kleine Schadsoftware zu installieren, die die sensiblen Daten aus dem Datenstrom zwischen Bank und Mitarbeiter-Arbeitsplatz mitliest.
Um Kosten zu sparen, verlagern viele Unternehmen ihre Rechenkapazitäten an externe IT-Dienstleistungsunternehmen. Da kommt schnell die Frage auf, „Wer überprüft diesen IT-Dienstleister?“. Leider befinden sich im Internet sensible Daten, die durch fehlerhafte oder absichtlich manipulierte Systeme abgefangen wurden. In einem Unternehmen ist es also jederzeit problemlos möglich eine *Man in the Middle* Attacke - ein anderer Rechner schaltet sich dazwischen - durchzuführen und so an sensible Dateien zu kommen. Das interessante an unserer Lösung ist, dass hier auf sogenannte HTTPS Stripping Attacken eingegangen wird. Eine Modifikation nur auf der Clientseite (Browser) ist nicht ausreichend, somit wurde eine Firefoxerweiterung und eine kleine Demonstrations-Webseite in PHP, welches die Attacken erkennen soll, realisiert.
Im folgenden Kapitel folgt die Erläuterung, wie die entwickelte Firefoxerweiterung sich verhält und welche Arbeit sie leistet. Danach folgt eine Beschreibung der Aufgaben, die der PHP Code auf dem Webserver zu leisten hat. Anschließend werden die Angriffe von Kapitel 3 auf die Lösung praktisch durchgeführt und die Funktionsweise der Software aufgezeigt.

4.1 Vorraussetzungen

Es wird davon ausgegangen, dass die üblichen als sensible Daten auch in diesem Beispiel als sensibel eingestuft werden. Hier ist es im Speziellen das Passwort (*pass*). Die Verschlüsselung der Daten wird mit Advanced Encryption Standard (AES) durchgeführt. Als Webserver Paket wurde XAMPP [18] ausgewählt. Die Firefoxerweiterung wurde komplett neu entwickelt; einige Grundlagen wurden aus der Erweiterung von SignatureCheck.org [2] entnommen.

4.1.1 Clientlösung: Firefoxerweiterung - Secure Online Banking

Die Firefoxerweiterung muss lokal auf jedem Rechner installiert werden, der das hier vorgestellte Verfahren unterstützen soll. Nach der Installation kann das Plugin direkt überprüfen, ob die Seiten, die es von Serverseiten unterstützten, gesichert übertragen werden. Die Funktionsweise der Firefoxerweiterung wird nun detailliert erläutert.

Damit die Firefoxerweiterung die Verbindung prüfen kann, müssen folgende Funktionen vorhanden sein:

1. Erkennung des Submit Events, welches beim Absenden eines Formulars ausgelöst wird,
2. Prüfung der Webseite auf die notwendigen Felder
3. Funktionen um Informationen sicher zu übertragen

Weiter verhindert die Erweiterung bei einem erkannten Angriff, dass vom Benutzer eingegebene Daten verschickt werden. Dabei wird eine Eigenschaft von HTML-Formularen ausgenutzt, die ein Submit-Event zum Absenden von Daten fordert. Die Firefoxerweiterung (Abb. 14) fängt dieses Submit ab und dadurch verhindert es auch das Absenden der Daten. Entsprechend kann hier die Erweiterung eingreifen und die sensiblen Daten des Benutzers schützen. Sollten diese in falsche

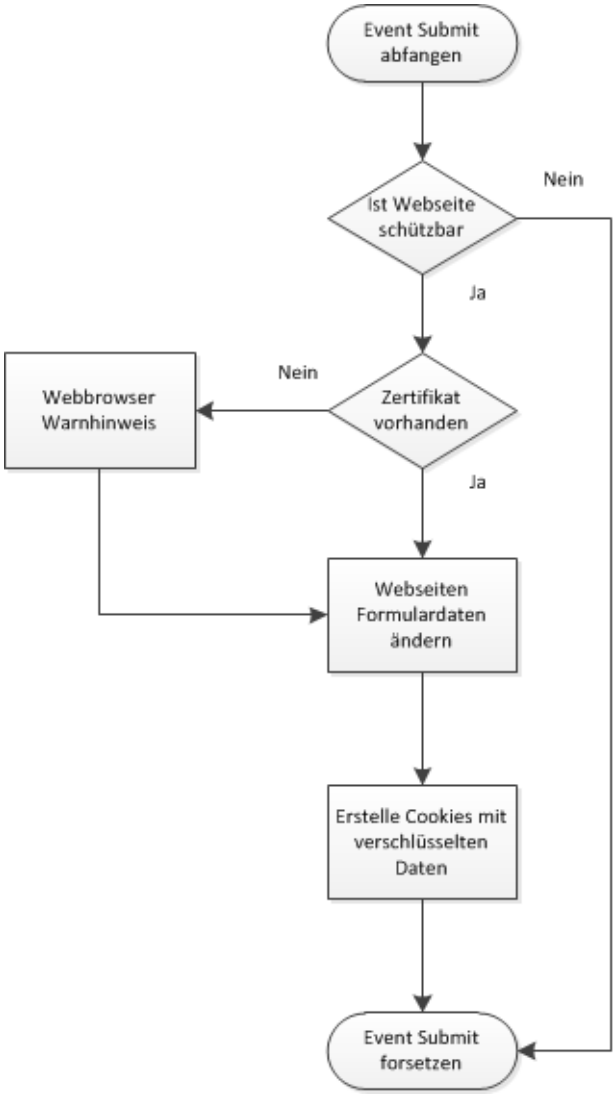
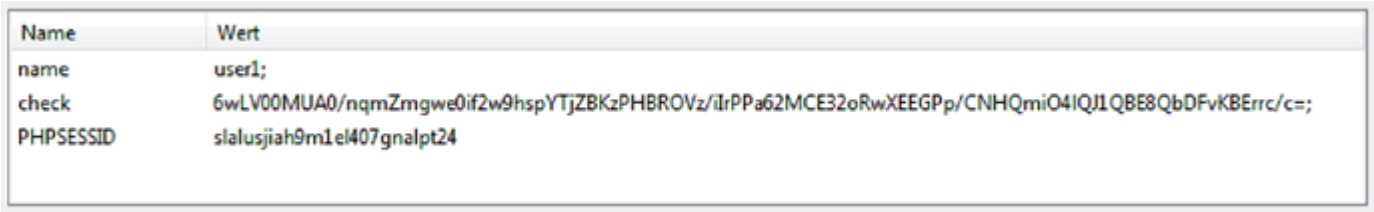


Abbildung 14: Ablaufdiagramm: Nachdem Event Submit gestartet wurde

Hände geraten, kann damit ein großer Schaden entstehen. Nach dem Abfangen des Events Submit erfolgt eine Webseitenidentifikation. Schlägt diese fehl, wird das Event Submit fortgesetzt, weil die Webseite nicht schützbar ist. Ansonsten wird geprüft, ob ein Zertifikat vorhanden ist. Ist kein gültiges Zertifikat vorhanden, wird ein Warnhinweis ausgegeben. Diese Meldung soll den Benutzer darüber informieren, dass seine sensiblen Daten nicht verschlüsselt übertragen werden.
Anschließend werden die Formulardaten von der Webseite ausgelesen. Diese enthält folgende Felder *name*, *pass* und $external\boldsymbol{v}al$. Das Feld *name* ist der Benutzername, das Feld *pass* ist für die Eingabe des Passwortes und das Feld $external\boldsymbol{v}al$ ist eine Zufallszahl, die vom Webserver generiert wurde. Mithilfe dieser Daten und der Information des Zertifikates kann die Verschlüsselung vollzogen werden. Es wird hierzu Advanced Encryption Standard (AES) mit 256 Bit verwendet und verschlüsselt folgenden String $AES_{Passwort}(Fingerprint, Zufallszahl)$. Falls kein Zertifikat vorhanden ist, wird anstelle des Fingerprints ein beliebiger Wert gesetzt.
Zusätzlich werden die Formulardaten verändert, um die sensiblen Daten zu schützen und der Angreifer trotz Abfangen der Daten keinen Erfolg beziffern kann. Das heißt, *pass* besitzt den Wert

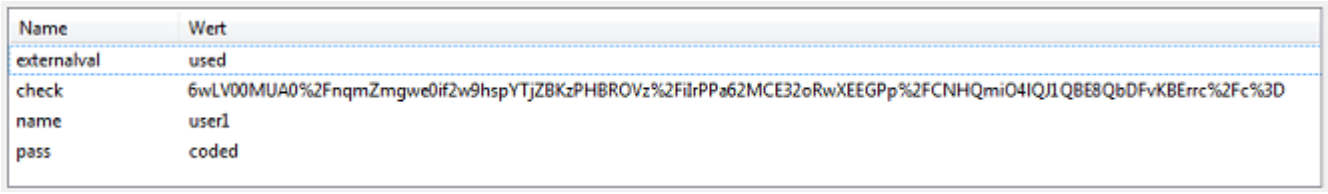
coded und in *externalval* wird *used* übergeben. Außerdem werden zwei Cookies (Abb. 15) *name* und *check* generiert und mit Base64 kodiert.



Name	Wert
name	user1;
check	6wLV00MUA0/nqmZmgwe0if2w9hspYTjZBKzPHBROVz/iIrPPa62MCE32oRwXEEGPp/CNHQmiO4IQJ1QBE8QbDFvKBErrc/c=;
PHPSESSID	slalusjiah9m1el407gnalpt24

Abbildung 15: Die angelegten Cookies

In *check* ist der oben generierte AES-String enthalten und in *name* der Benutzername. Passwort und Benutzername werden im Firefoxplugin gespeichert. Schlussendlich wird das Event Submit fortgesetzt.



Name	Wert
externalval	used
check	6wLV00MUA0%2FnqmZmgwe0if2w9hspYTjZBKzPHBROVz%2FiIrPPa62MCE32oRwXEEGPp%2FCNHQmiO4IQJ1QBE8QbDFvKBErrc%2Fc%3D
name	user1
pass	coded

Abbildung 16: Auszug wie die POST Daten übermittelt werden

Der Request, der anschließend von dem Webbrowser an den Webserver gesendet wird, ist in Abbildung 16 näher zu betrachten.

Abbildung 17 zeigt, dass nach einer Firefox Statusänderung, wie z.B Webseitenwechsel oder -neuladen, eine interne Funktion aufgerufen wird. Es wird nicht wie beim Login ein Submit Event benötigt. Zuerst wird überprüft, ob es sich um unsere schützbare Webseite handelt. Nur bei erfolgreicher Verifizierung erfolgt eine Zertifikatskontrolle. Kommt es hierbei zu einer negativen Bedingung, weil kein Zertifikat vorhanden ist, wird ein Warnhinweis (Abb. 18) im Webbrowser ausgegeben. Die Aufgabe des Warnhinweises ist es, dem Benutzer mitzuteilen, dass seine Daten unverschlüsselt übertragen werden. Trifft der Benutzer dennoch die Entscheidung seine Daten abzuschicken, wurde dem Benutzer durch eine zusätzliche Meldung bewusst gemacht, dass geheime Daten, wie Passwörter, TAN, et cetera, im Klartext übertragen werden und mitgelesen werden können.
Die Erweiterung besitzt nun die Fähigkeit einen Datenaustausch zu ermöglichen, indem die Daten, die auf der Webseite eingegeben wurden, mit in die Verschlüsselung einzubinden und anschließend in einem Cookie zu verpacken. Vor dem Absenden des Requests, welches nicht durch die Erweiterung, sondern durch den Webbrowser durchgeführt wird, werden die Daten auf der Webseite unkenntlich gemacht, sodass im Klartext nur die Verschlüsselung und Benutzername als Cookie gesendet werden.
Eine Verschlüsselung könnte wie folgt ausschauen:
$AES_{Passwort}(Fingerprint, Zufallszahl, Daten)$

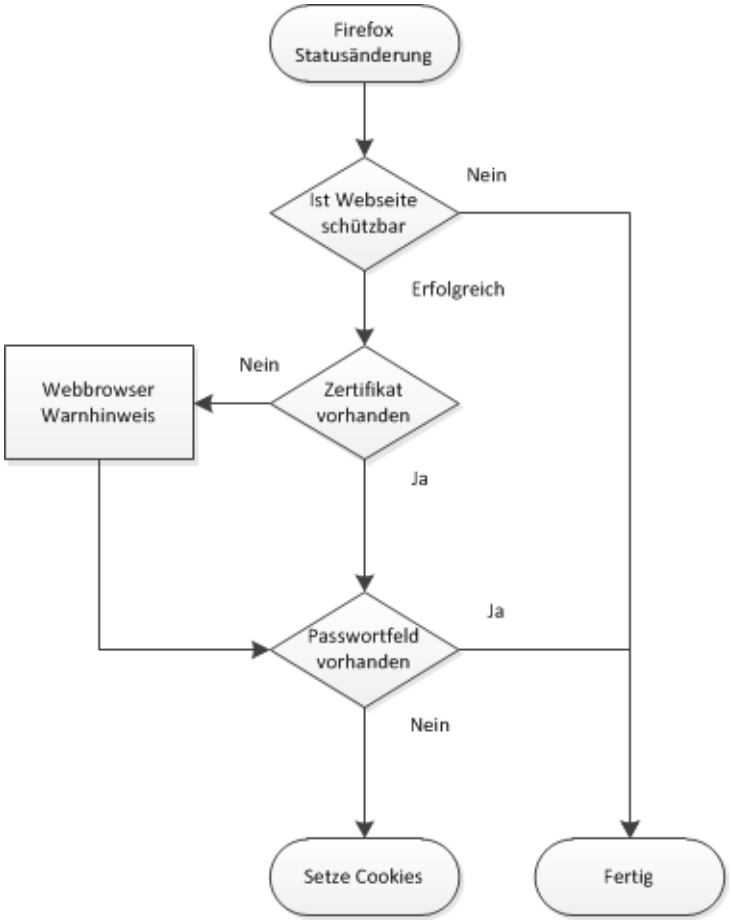


Abbildung 17: Ablaufdiagramm: Nach dem eine Webseite geladen wurde.

Abbildung 18: Warnhinweis im Firefox

4.1.2 Serverlösung - PHP Skript

Die Webseite muss um den zusätzlichen PHP Code erweitert werden. In der Abbildung 19 werden die Aufgaben des Web-Servers veranschaulicht. Auf der Loginseite wird das Formular um ein Feld $external val$ erweitert. Dieses Feld enthält eine vom Webserver generierte Zufallszahl. Jedes Mal, wenn die Loginseite aufgerufen wird, wird ein neuer Nonce erzeugt, in eine Datenbank gespeichert und mit einem Zeitstempel versehen. Der Zufallswert hat die Aufgabe Replay Angriffe zu verhindern. Nachdem der Request gesendet wurde, folgt eine Loginüberprüfung. Dabei wird zuerst geprüft, ob die Cookies übertragen sind, andernfalls wird die Anmeldung verweigert. Um sich erfolgreich Anzumelden, wird geprüft, ob der Benutzername in der Datenbank gespeichert ist. Mithilfe des Benutzernames kann das Passwort ermittelt werden, was dazu benötigt wird, um den AES String, der als Cookie übertragen wurde, zu entschlüsseln. Zum Schluss wird kontrolliert, ob der Fingerprint von beiden Seiten übereinstimmt und die Gültigkeit der Zufallszahl nicht abgelaufen ist. Ist der Fingerprint ungleich, wird auf eine externe Seite (Abb. 20) weitergeleitet, die dem Benutzer verdeutlicht, dass wahrscheinlich ein Angriff mit falschem Zertifikat durchgeführt wurde. War die Nonce ungültig, dann wird der Login ebenfalls verweigert, da wahrscheinlich ein

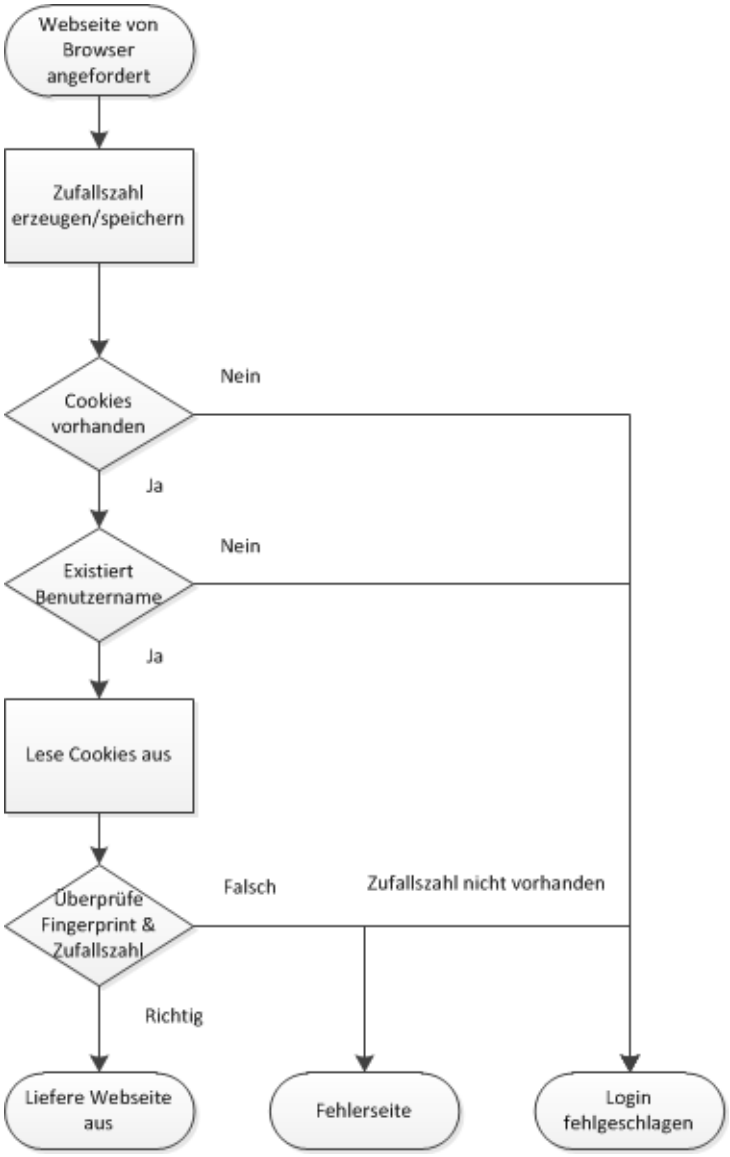


Abbildung 19: Ablaufdiagramm: Vorgang des Web-Servers

Replayangriff versucht wurde.
Sind alle Überprüfungen bestanden, wird der Login gewährt.

Achtung!

Wir möchten Sie darauf hinweisen, dass der Fingerprint des Zertifikates falsch ist.

Die angezeigte Webseite stammt nicht von uns. Versuchen Sie es hier erneut.

Abbildung 20: Ausgabe des Webservers: Zertifikat ist falsch

4.2 Angriffszenario

Zu Testzwecken wurde ein Anmeldeverfahren (Abb. 21) entwickelt, wie es häufig bei Webseiten für OnlineBanking verwendet wird. Auf dieser Plattform werden die Angriffe jeweils mit aktivierten und deaktivierten Firefoxerweiterung gezeigt. Wobei bereits der spezifische PHP Code bei der Gegenstelle mit implementiert wurde.

Abbildung 21: Webseitendarstellung eines Logins

4.2.1 SSLStrip ohne Firefoxerweiterung

Hier wird ein Angriff ohne aktive Firefoxerweiterung durchgeführt. In der Browseradresszeile wird ein HTTPS Link eingegeben und durch SSLStrip in ein HTTP Link umgeleitet. Die daraufhin aufgerufene Webseite zeigt keine Unterschiede zu der sicheren Webseite.

Abbildung 22: Angriff SSLStrip ohne Firefoxerweiterung

Das einzige Erkennungsmerkmal ist die geänderte Adresszeile (Abb. 22). Der Webbrowser warnt den Benutzer nicht, dass die Adresszeile verändert wurde und man sich auf keiner HTTPS Webseite befindet. Versucht man eine Anmeldung durchzuführen, hat der Angreifer sein Ziel erreicht und kann die sensiblen Daten (Abb. 23) einfach mitsniffen.

```
2012-01-14 12:34:17,931 POST Data (localhost):
externalval=835&name=user1&pass=test&login=Login
```

Abbildung 23: Angriff SSLStrip: Daten abgehört ohne Firefoxerweiterung

Zwar erfolgt bei unserer Lösung keine erfolgreiche Anmeldung, da die PHP Skript Erweiterung auf der Serverseite den Fingerprint überprüft und er nicht mitgesendet wurde. Doch bei Webseiten ohne diese Überprüfung bzw. ohne dem spezifischen PHP Code auf der Webserverseite würde die Anmeldung gelingen.

4.2.2 SSLStrip mit Firefoxerweiterung

Als nächstes folgt der SSLStrip Angriff mit aktivierter Firefoxerweiterung. Die Firefoxerweiterung erkennt im Gegensatz zum Firefox Webbrowser, dass sich die Adresszeile geändert hat und das keine sichere Verbindung besteht. Dies wird durch einen Warnhinweis (Abb. 24) angezeigt. Das

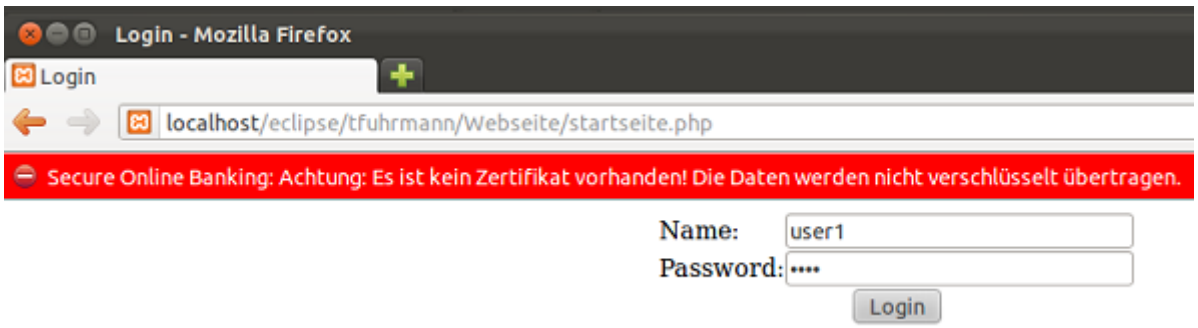


Abbildung 24: Angriff SSLStrip mit Firefoxerweiterung

Submit Event wird abgefangen und trotz, dass kein Zertifikat existiert, werden die Formulardaten vor dem Submit verändert, sodass sie für einen Angreifer wertlos sind. Das heißt, dass das Passwortfeld auf den Wert *used* gesetzt wird. Obwohl der Angreifer einen erfolgreichen Angriff vollzogen hat, bringen ihm die abgefangenen Daten (Abb. 25) nichts, da der Fingerprint fehlt und somit die Anfrage vom Webserver abgewiesen wird.

```
2012-01-15 11:55:55,044 POST Data (localhost):
externalval=used&
check=cQEcvrqwEk%2B7XF5wjPMCvsfgIACNcwYM6twEEzORVGCW3%2FFCe1XRXNIDmbQb0Y1C&
name=user1&
pass=coded&
login=Login
```

Abbildung 25: Auszug aus der Logdatei des SSLStrip Angriffes

4.2.3 SSL Man in the middle Proxy - ohne Firefoxerweiterung

Auf die gleiche Testwebseite wird der zweite Angriff SSL Man in die Middle Proxy (3.2) durchgeführt. Nach dem Aufruf des HTTPS Link werde ich von Firefox aufgefordert eine Ausnahme (Abb. 26) hinzufügen, da das Zertifikat nicht vertrauenswürdig ist. Firefox erkennt, dass das Zertifikat von keinem bekannten und vertrauenswürdigen Zertifikatsaussteller ausgestellt wurde. So eine Nachricht ist beim Webbrowser häufig zu sehen, da es Organisationen gibt, die nicht die finanziellen Mittel besitzen sich bei einer vertrauenswürdigen Zertifikatsstelle zertifizieren zu lassen. Ein Beispiel dafür ist die Webseite der TU Darmstadt, bei der ebenfalls eine Ausnahmebestätigung nötig ist.

Fügt man einmalig diese Ausnahme hinzu, wird man auf die Webseite weitergeleitet und es erscheint kein weiterer Warnhinweis mehr. Für den Benutzer besteht optisch kein Unterschied zu der originalen Webseite (Abb. 21). Erst bei genauer Prüfung wird erkannt, dass das Zertifikat gefälscht wurde. (Abb. 28, 27).

Bei dem Vergleich der Zertifikate erkennt man, dass diese für die gleiche Organisation ausgestellt wurden. Dennoch besitzen die Zertifikate verschiedene Zertifizierungsstellen, Fingerabdrücke und

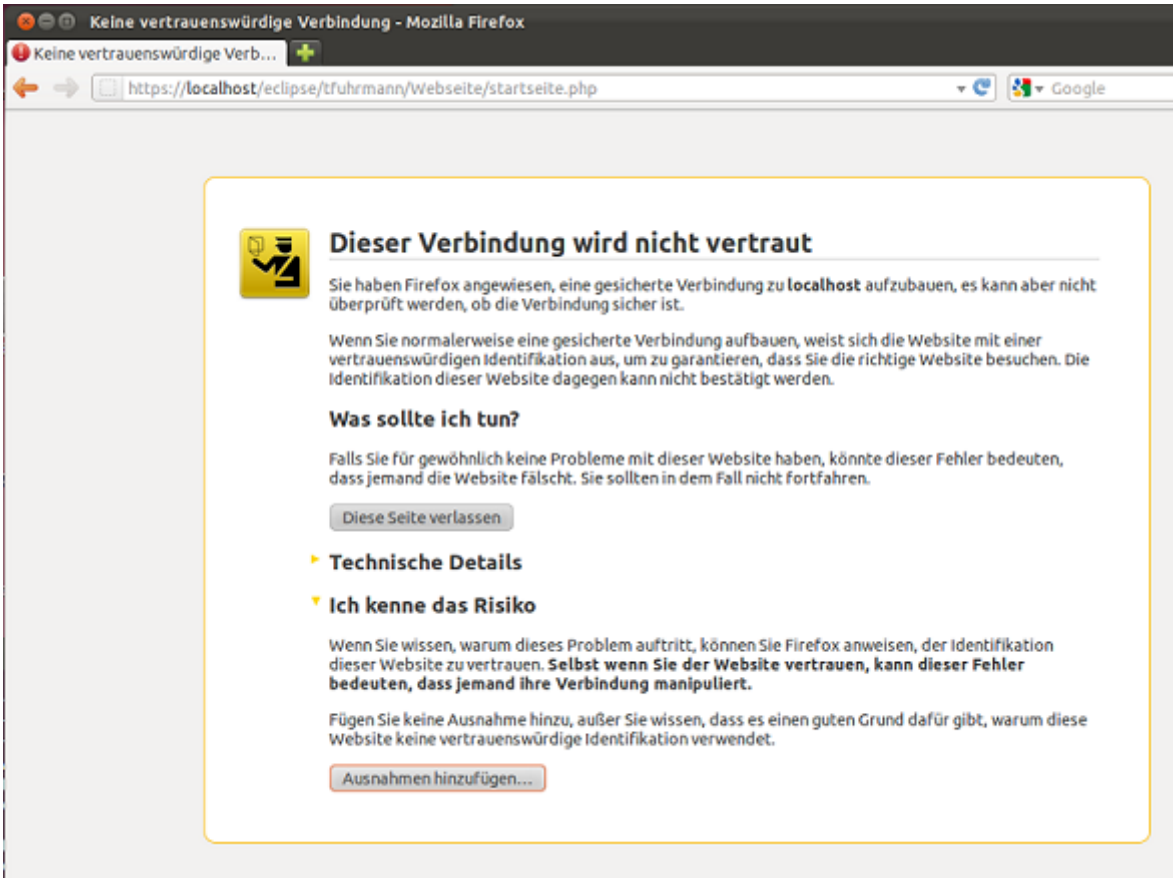


Abbildung 26: Firefox Warnung bei dem Angriff SSL Man in the middle Proxy

eine Validität. Das hat zur Folge, dass trotz einer sicheren HTTPS Verbindung, der Angreifer die Möglichkeit besitzt die Daten auszulesen.

Wird das Formular versendet, kann der Angreifer die Daten (Abb.29) abfangen und entschlüsseln. Dennoch schlägt die Anmeldung beim Webserver fehl, weil die weitergeleiteten Daten vom SSL Proxy nicht den Kriterien des Webservers entsprechen.

4.2.4 SSL Man in the middle Proxy - mit Firefoxerweiterung

Diesmal erfolgt der Angriff mit aktivem Firefoxplugin. Nach Aufruf der Webseite ist weiterhin kein Unterschied zu der originalen Webseite zu erkennen. Versucht der Benutzer sich anzumelden, wird die Firefoxerweiterung aktiv und fängt das Submit Event ab. Zusätzlich werden die Formulardaten geändert und die Erweiterung liest den Fingerprint des Zertifikates aus. In diesem Fall wäre es das Zertifikat (Abb. 27) das von dem Angreifer generiert wurde. Anhand dieser Information wird die Verschlüsselung $AES_Passwort(Fingerprint, Zufallszahl)$ erzeugt. Obwohl der Angreifer den Datenverkehr mitlesen kann, kann er mit den abgefangenen Daten (Abb. 30) nichts anfangen.

Nachdem der Request gesendet wurde, erkennt der Webserver, dass die angezeigte Webseite ein falsches Zertifikat besaß und teilt dies dem Benutzer durch eine Fehlerseite (Abb. 20) mit. Damit wurde trotz des Angriffes das Anmelden untersagt und zusätzlich wurde der Benutzer darauf hingewiesen, dass das Zertifikat eine Fälschung ist.

Abbildung 27: Ausgestelltes Zertifikat von dem Angriff SSL Man in the middle Proxy

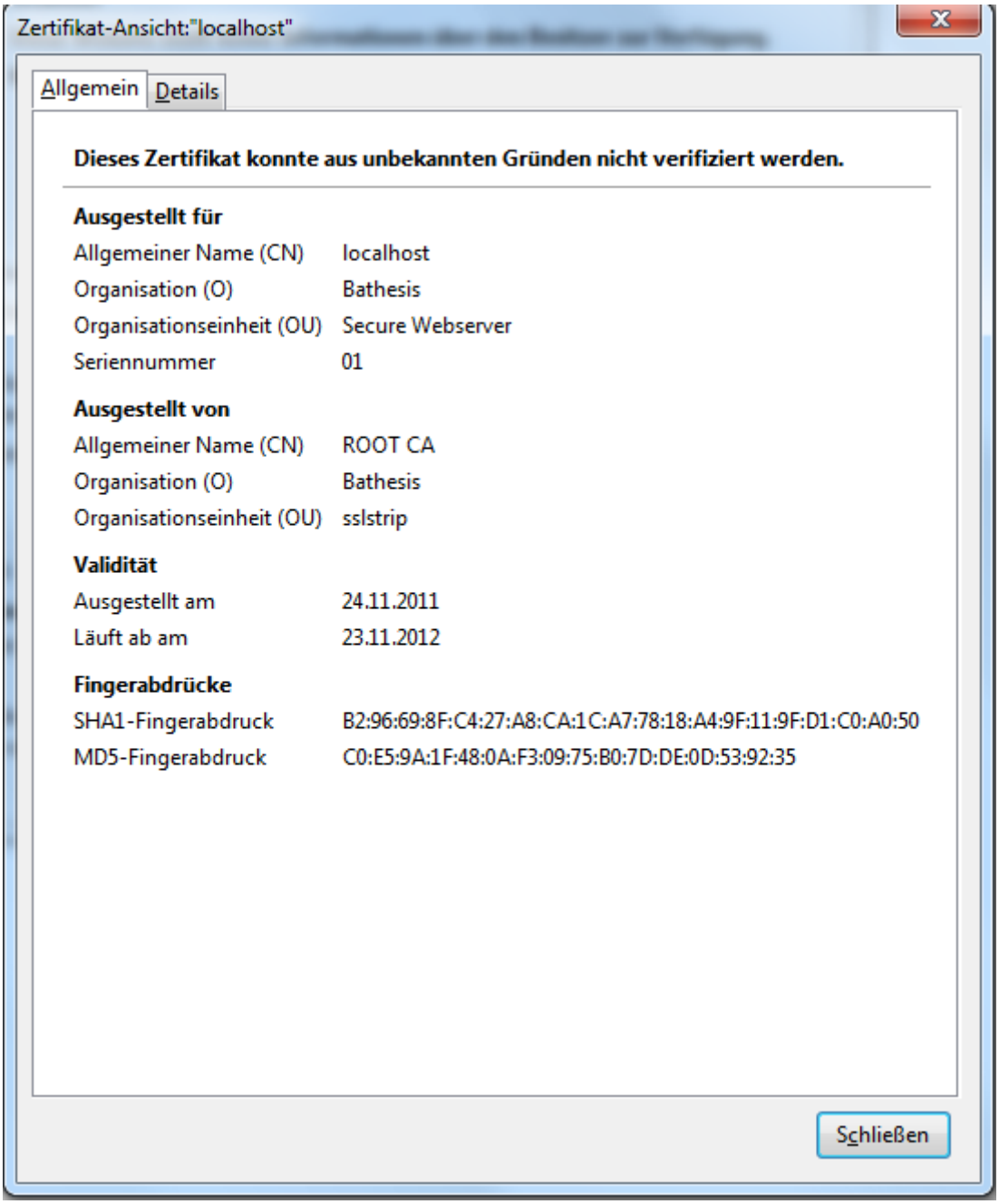


Abbildung 28: Originales Zertifikat der Webseite

```
------ localhost:45196->localhost:443 ------
POST /eclipse/tfuhrmann/Webseite/login.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Ubuntu; X11; Linux x86_64; rv:9.0.1) Gecko/20100101 Firefox/9.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Referer: https://localhost/eclipse/tfuhrmann/Webseite/startseite.php
Cookie: PHPSESSID=sp3ta5tnqi0s6drr54s4lv22p5
Content-Type: application/x-www-form-urlencoded
Content-Length: 47

externalval=76&name=user1&pass=test&login=Login
```

Abbildung 29: Auszug des abgehörten Datenverkehrs des SSL Man in the middle Angriffes ohne Firefoxerweiterung

```
------ localhost:38980->localhost:443 ------
POST /eclipse/tfuhrmann/Webseite/login.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Ubuntu; X11; Linux x86_64; rv:9.0.1) Gecko/20100101 Firefox/9.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Referer: https://localhost/eclipse/tfuhrmann/Webseite/startseite.php
Cookie: name=user1;
        check=VFFKekYzcG1FVSthRUdIUTBYbFc0YnhqdU44QkFaMWcxbGU3bmVnbnM4S3pnTFlHNmVJZ25xeTFBZm
        PHPSESSID=sp3ta5tnqi0s6drr54s4lv22p5
Content-Type: application/x-www-form-urlencoded
Content-Length: 157

externalval=used&check=TQJzF3pmEU%2BaEGHQ0XlW4bxjuN8BAZ1g1le7negns8KzgLYG6eIgnqy1AfmcBZNUUF4
```

Abbildung 30: Auszug des abgehörten Datenverkehrs des SSL Man in the middle Angriffes mit Firefoxerweiterung

5 Fazit

Der praktische und der theoretische Teil der Arbeit haben gezeigt, dass die HTTPS Stripping Angriffe vereitelbar sind. Dies ist nur mit Hilfe einer speziell entwickelten Erweiterung für den Browser möglich, der mit bereitgestellten Werten des Webservers interagiert. In dieser Arbeit wurde sich ausschließlich dem Firefox Browser gewidmet.
Es fällt auf, dass der Benutzer selbst bei einer sicher geglaubten HTTPS Verbindung nicht vor HTTPS Attacken geschützt wird. Der Benutzer trägt also selber die Verantwortung und muss die teilweise minimalen Unterschiede erkennen und entscheiden, ob es sich um die originale Webseite handelt. Dazu geben die aktuellen Webbrowser und Webserver keinerlei Entscheidungshilfe. Die in dieser Arbeit vorgestellten Konzepte sollen beiden Parteien helfen die richtige Entscheidung zu treffen und schließlich den Benutzer vor dem Missbrauch seiner hochsensiblen Daten zu schützen. Weiterhin sind Schwachstellen der Browser aufgefallen, die unnatürliches Verhalten nicht unterbindet und ohne weiteres die empfangenen Daten dem Benutzer präsentiert. Besonders ist aufgefallen, dass sensible Daten einfach eingeben und abgesendet werden können, obwohl keine sichere Verbindung vorhanden ist. Vor allem Webserver, die OnlineBanking bereitstellen, schützen oder gar erkennen HTTPS Stripping Attacken nicht. Der Benutzer muss alleine die Attacke erkennen. Ihm wird eindeutig zu viel abverlangt, vor allem da selbst Experten es schwer haben die Unterschiede zwischen originalen oder gefälschten Webseiten aufzudecken, da homographische Angriffe immer besser werden.
Der Benutzer wird es weiterhin im World Wide Web schwer haben zu unterscheiden, ob es sich um die originale Webseite handelt und ob die Eingabe sensibler Daten ohne Bedenken möglich ist. Ich hoffe, ich trage mit dieser Arbeit dazu bei, dass es für Benutzer in Zukunft einfacher wird zu entscheiden, ob die aufgerufene Webseite eine vertrauenswürdige Webseite und ob eine sichere Verbindung vorhanden ist. Zudem erhoffe ich mir, dass ich mit dieser Arbeit erste Ansätze liefern konnte, die HTTPS Stripping Attacken unbrauchbar machen.

5.1 Ausblick

Es wäre interessant die Erweiterung für andere beliebte Webbrowser zu entwickeln, um mehr Benutzer damit zu erreichen. Die Erweiterung hat keine optimale Sicherheitslösung zum Zwischenspeichern des Benutzernames und des Passwortes. Dies könnte zum Beispiel durch Auslagerung der sensiblen Daten auf dem Rechner des Benutzers realisiert werden. Des Weiteren müsste die Erweiterung bezüglich dem weiteren Datenaustausch nach dem Login entwickelt werden. Zusätzlich sollte für den Webserver der PHP Code verbessert und angepasst werden. Erst dann ist es auch für Firmen interessant. Des Weiteren könnte man eine Studie aufsetzen mit dem Ziel nachzuforschen, welche Signale und Hinweise ein Webbrowser anzeigen muss, so dass ein Benutzer eine unverschlüsselte Verbindung wahrnimmt.

5.2 Dankessagung, Arbeitsgruppe

Ich möchte mich bei Marco Ghiglieri bedanken, denn ohne sein engagiertes Mitwirken und Betreuen wäre diese Arbeit gar nicht erst entstanden.
Des Weiteren möchte ich mich bei meiner Familie für das Korrekturlesen bedanken, sowie meiner Freundin Lea, die mir dabei half immer hochmotiviert an der Thesis zu arbeiten.
Ein großes Dankeschön gehört ebenfalls meinem Komilitonen Christopher Liebchen, der mir mit hilfreichen Ratschlägen und Korrekturlesen zur Seite stand.

Schließlich möchte ich bei SIT Research Group bedanken und vorstellen, wo ich meine Bachelorarbeit geschrieben habe. Die SIT Research Group arbeitet unter der Leitung von Prof. Dr. Michael Waidner für die Informatikabteilung der Technischen Universität Darmstadt. Ihre Forschungsgebiete sind Security and Privacy in Cloud Computing, Secure Engineering, Privacy and Identity Management und Cryptography and Cryptographic Protocols. [25]

Code der Firefoxerweiterung

basis.js

```
1 const STATE_START = Components.interfaces.nsIWebProgressListener.STATE_START;
2 const STATE_STOP = Components.interfaces.nsIWebProgressListener.STATE_STOP;
3
4 Components.utils.import("resource://gre/modules/NetUtil.jsm");
5
6 // Statusvariabel um den Ladestatus widerzuspiegeln
7 var load=false;
8
9 // Globale Variabel die Pin(Passwort) und Benutzername speichert
10 var g_PIN="";
11 var g_username="";
12
13 var myListener =
14 {
15   QueryInterface: function(aIID)
16   {
17    if (aIID.equals(Components.interfaces.nsIWebProgressListener) ||
18        aIID.equals(Components.interfaces.nsISupportsWeakReference) ||
19         aIID.equals(Components.interfaces.nsIObserver) ||
20        aIID.equals(Components.interfaces.nsISupports))
21      return this;
22    throw Components.results.NS_NOINTERFACE;
23   },
24
25   onStateChange: function(aWebProgress, aRequest, aFlag, aStatus)
26   {
27    // Wird bei Auruf von Tab/Fenster aufgerufen
28    if(aFlag & STATE_START)
29    {
30      // Dieses Ereignis wird aufgerufen, wenn das Ladeevent initiiert ist
31    }
32    if(aFlag & STATE_STOP)
33    {
34
35      if(!load)
36      {
37         checkWebsite();
38      }
39      load=true;
40
41    }
42   },
43
44   onLocationChange: function(aProgress, aRequest, aURI)
45   {
46    load=false;
47    getCertificateInfo(); // checkt ob Zertifikat vorhanden ist und warnt einen,
   wenn nicht
48   },
49
50   onProgressChange: function(aWebProgress, aRequest, curSelf, maxSelf, curTot,
   maxTot) { },
51   onStatusChange: function(aWebProgress, aRequest, aStatus, aMessage) { },
52   onSecurityChange: function(aWebProgress, aRequest, aState) { }
53 }
54
55
56
57 function checkWebsite()
58 {
59     // Hol mir den Inhalt des derzeitigen angezeigten Fenster
60     var win = window.content;
61
62     // besitzt das Dokument eine externalval Variable
```


basis.js

```
63     var ub=win.document.getElementById("externalval");
64
65     //Wenn es existiert dann...
66     if(ub!=null) {
67
68         //Zurerst SSL kontrollieren
69          var cert=getCertificateInfo();
70
71         var getPin = win.document.getElementById("pass");
72
73         //Ist es ein Form oder nicht
74         if(getPin!=null)
75         {
76             //Ja, dann mach erstmal nichts... musst auf das submit event warten
77
78         }
79         else
80         {
81             //Keine Passwortübertragung kein Formular normaler Datenverkehr kann
   hier geregelt werden
82
83             //Cookie setzen check und name
84             //cert=getCertificateInfo();
85             if(cert != null)
86                 var ciphertext = aesfkt(ub.value,g_PIN, cert.sha1Fingerprint);
87             else
88                 var ciphertext = aesfkt(ub.value,g_PIN,"");
89
90             // cookies erzeugen
91             var b = Base64.encode(ciphertext);
92             win.document.cookie="name=" +g_username;
93             win.document.cookie="check=" +b;
94
95         }
96     }
97
98     return;
99 }
100
101 /**
102 *   Wird ein Submit gedrückt wird diese Funktion aufgerufen.
103 *   Sie verändert den Request, indem ein neues Feld check angelegt wird. Des
    Weiteren wird Password und
104 *   Zufallswert unkenntlich gemacht. Zusätzlich werden zwei Cookies check & name
    erstellt.
105 */
106 function eventSubmit(aEvent){
107
108         // Hol mir den Inhalt des derzeitigen angezeigten Fenster
109         var win = window.content;
110
111         var ub=win.document.getElementById("externalval");
112
113         // Ist externalval vorhanden
114         if(ub!=null) {
115
116             // Hole mir das Zertifikat
117             var cert = getCertificateInfo();
118
119             // Password auslesen
120             var getPin = win.document.getElementById("pass");
121
122             if(cert != null)
123                 var ciphertext = aesfkt(ub.value,getPin.value,
```

```
    cert.sha1Fingerprint);
124             else
125                 var ciphertext = aesfkt(ub.value,getPin.value,"");
126
127
128             var evl=win.document.createElement("input");
129             evl.setAttribute("type", "hidden");
130             evl.setAttribute("name", "check");
131             evl.setAttribute("id", "check");
132             evl.setAttribute("value", ciphertext);
133
134             //Cookies werden erzeugt
135             var b = Base64.encode(ciphertext);
136             win.document.cookie="name=" +
    win.document.getElementById('name').value;
137             win.document.cookie="check=" + b;
138
139             win.document.getElementById("externalval").appendChild(evl);
140             var form=win.document.getElementById("formaction");
141
142
143             g_PIN=getPin.value;
144             g_username=win.document.getElementById("name").value;
145
146             //Verschleiern vom originalen Passwort
147             getPin.value="coded";
148             //Verschleiern vom originalen Zufallswert
149             win.document.getElementById("externalval").value="used";
150
151
152             form.submit();
153
154         }
155 }
156
157 /**
158 * Holt sich das Zertifikat
159 * Beim Fehler wirft es ein Warnhinweis
160 * Rückgabe: Zertifikat
161 */
162 function getCertificateInfo()
163 {
164     var cert=null;
165
166     try {
167         var ui = gBrowser.securityUI;
168         if (ui.state==4){ // Kein Zertifikat vorhanden!
169             firefoxalert("Achtung: Es ist kein Zertifikat vorhanden! Die Daten
    werden nicht verschlüsselt übertragen.", "high");
170             cert = null;
171         } else {
172             sp = ui.QueryInterface(Components.interfaces.nsISSLStatusProvider);
173             status = sp.SSLStatus;
174             status = status.QueryInterface(Components.interfaces.nsISSLStatus);
175             cert = status.serverCert;
176         }
177     } catch(e) { // Kein Zertifikat auslesbar
178         cert = null;
179         firefoxalert("Achtung: Es ist kein Zertifikat vorhanden! Die Daten werden
    nicht verschlüsselt übertragen.", "high");
180     }
181
182     return cert;
183 }
```



```
184
185 /**
186 *  Ausgabe eines Warnhinweis im Firefox
187 */
188 function firefoxalert(message,messagepriority){
189
190     var nb = gBrowser.getNotificationBox();
191     var n = nb.getNotificationWithValue('warning');
192     if(n) {
193         removeAllNotifications(true);
194     }
195     var buttons = [];
196     if (messagepriority=="medium"){
197         priority = nb.PRIORITY_CRITICAL_LOW;
198     } else if (messagepriority=="low") {
199         priority = nb.PRIORITY_INFO_LOW;
200     } else {
201         priority = nb.PRIORITY_CRITICAL_BLOCK;
202     }
203     nb.appendNotification("Secure Online Banking: "+message, 'warning',
204                           '',
205                           priority, buttons);
206     return;
207 }
208
209 /**
210 * Verschlüsselung mit AES: AES_password(Fingerprint,Zufallszahl)
211 * Rückgabe ist der Ciphertext
212 */
213 function aesfkt( rdm , pin, fp){
214
215     var klartext;
216     var ciphertext = "";
217
218     if(fp == "")
219     {
220         klartext = "baskfdbsafkldsf" + "," + "kein FP darf nicht gehen";
221
222         ciphertext = Aes.Ctr.encrypt(klartext,pin,256);
223     }
224     else{
225         klartext = fp + "," + rdm;
226
227         ciphertext = Aes.Ctr.encrypt(klartext,pin,256);
228
229     }
230
231     return ciphertext;
232
233 }
234
235
236
237 // Wenn Submit gedrückt wurde, spricht er zur Funktion eventSubmit
238 window.addEventListener("submit", eventSubmit, true);
239
240 // Firefox events
241 gBrowser.addProgressListener(myListener);
242
243
244
245
246
247
```

Code zu Erzeugung und Speicherung der Nonce

random_gen.php

```php
<?php
    //Diese Datei kann überall dort eingebunden werden, wo ein Random benötigt wird.
    include 'config.php';

    $rdm = rand(0, 1000); //erzeuge Zufallszahl

    //Alle Randoms die älter als ein Tag sind löschen
    $query="DELETE FROM `requestrdm` WHERE timestamp <= ADDDATE(CURRENT_TIMESTAMP , INTERVAL -24 HOUR)";
    mysql_query($query,$conn);

    // Testen, dass das Rdm nicht 2mal vorkommt, ansonsten neu generieren
    $query ="SELECT Random From requestrdm WHERE Random = ".$rdm.";";  //ob die Zufallszahl schon in der DB vorhanden ist
    $result=mysql_query($query,$conn);

    if ($result) {
        if ($count_result=mysql_affected_rows() > 0) {
            while ($line=mysql_fetch_row($result)) {
                // Fehlerbehandliung, da es schon einen Datensatz mit den entsprechenden Zufallszahl gibt
                $rdm = rand(0, 1000);
                $query = "SELECT Random From requestrdm WHERE Random = '".$rdm."';";
                $result=mysql_query($query,$conn);
            }

        }
        // In die Datenbank schreiben
        $result=mysql_query("INSERT INTO requestrdm (Random,timestamp) VALUES ('".$rdm."',CURRENT_TIMESTAMP);");

        if (!$result)// Fehler bei der DB
            die('Ungültige Anfrage: ' .mysql_error());
    }
    else {
    // Fehler bei der DB
        die('Ungültige Anfrage: ' .mysql_error());
    }

    mysql_close($conn);
?>
```

Überprüfung der Anmeldung

_login.php

```php
<?php
session_start();
 echo "warst du hier _login";
//Start Class
require('login_pw.php');

if(isset($_SESSION['count'])){
    $_SESSION['count'] = $_SESSION['count'] + 1;
}
else
    $_SESSION['count'] = 1;

$login = new login_class;

$need_login = FALSE;

$test = isset($_POST['pass']);

echo $test;

//$login->users = $users;

//$login->username = $_POST['name'];
// Verifzierungs fehlt


//Logged In
// Fehlermeldung ist abzufangen
if (isset($_POST['name']) && isset($_POST['pass']) ) {
    echo "test";
    $name = $_POST['name'];

    if(in_array($name , $login->users)){
        echo "name erkannt gefunden";
     if( $login->users[$name] == $_POST['pass']){
        $_SESSION['username'] = true;
         $login->username = $name;
        $need_login = FALSE;
     }
    }
    //else
    //  $need_login = TRUE;

}
else
    $need_login = true;

    //Login Page
    if ($need_login && !isset($_SESSION['username'])) {
        require('startseite.php');
        exit();
    }


?>
```

Anmeldeseite

login.php

```php
<?php
require 'Aes.php';
require 'AesCtr.php';
include 'random_gen.php';
include 'config.php';
require 'login_pw.php';

session_start();

$aes = new AesCtr;
$login = new login_class;

$need_login = FALSE;

if(isset($_COOKIE['check'])){
    if(isset($_COOKIE['name'])){

        $name = $_COOKIE['name']; //Name speichern
        foreach($login->users as $u => $p){ // Name überprüfen
            if(strcmp($name, $u) == 0){
                $login->username = $name;
                break;
            }
        }
        if($login->username == ""){
            session_destroy();
            header('LOCATION: startseite.php?err=11');
            break;
        }

        $check = base64_decode($_COOKIE['check']); //decodieren

        $plaintext = $aes ->decrypt($check,$login ->users[$name],256); // check entschlüsseln
        list($fingerprint, $checkRdm) = explode(",", $plaintext);   //string teilen


        if(ctype_digit($checkRdm)){
            // Datenbankabfrage ob die Zufallszahl enthalten ist.
            $result = mysql_query("SELECT Random From requestrdm WHERE Random = '".$checkRdm."';");
            if($result){
                if ($count_result=mysql_affected_rows() > 0){ // gefunden
                    // Random Wert löschen
                    mysql_query("DELETE FROM requestrdm WHERE Random = '".$checkRdm."';");
                }
                else{ //nicht vorhanden mehr
                    header('LOCATION: startseite.php?err=12');
                    $need_login = true;
                    break;
                }
            }
            else{
                die('Ungültige Anfrage: ' . mysql_error());
            }

            mysql_close($conn);
        }
        else{ // Split nicht erfolgreich Passwort muss falsch sein oder Fehler in der Übertragung
            header('LOCATION: startseite.php?err=1');
            $need_login = true;
            break;
```


login.php

```php
        }

        $equal = strcmp($fingerprint, $FPRINT); //Fingerprint überprüfen
        if($equal != 0){ // Fingerprint falsch;
            session_destroy();
            header('LOCATION: fehler.php');
            break;
        }

        $need_login = false;
    }
    else
        $need_login = true;
}
else
    $need_login = true;


//Login fehlgeschlagen
if ($need_login) {
    session_destroy();
    header('LOCATION: startseite.php?err=14');
}

?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2" />
<title>Secure PHP Login without Database</title>

</head>

<body>

<strong>Willkommen <?php echo $login->username; ?>,</strong><br />
Vielen Dank fürs Einloggen<br /><br />

<input type="hidden" name="externalval" id="externalval" value="<?php echo $rdm; ?>"></input>

<a href="logout.php">Logout</a>

</body>
</html>
```

Glossar

Advanced Encryption Standard

Das Advanced Encryption Standard (AES) ist eine symmetrische Blockchiffre mit fester Blocklänge von 128 Bit. Das bedeutet, dass bevor die Daten verschlüsselt werden sie zunächst in Blöcke unterteilt werden, damit auch längere Blöcke verschlüsselt werden können. Es ist der Nachfolger des Data Enryption Standard (DES) und ist sicherer als das Tripel-DES Verfahren. Es ist sicherer als das Tripel-DES. Es erlaubt variable Schlüsselänge von 128-, 192- und 256 Bit Schlüssel. Dabei wird jeder Klartextblock in mehreren Runden mit einer sich wiederholenden Abfolge von Funktionen bearbeitet. Die Anzahl der durchzuführenden Runden ist von der Schlüsselänge abhängig.[6, 24]

Base64

Base64 ist ein abgespeckter ASCII-Zeichensatz, der aus 64 Buchstaben, Ziffern und Sonderzeichen besteht. Er wird für die Codierung von Binärdaten zur Übertragung von E-Mails benutzt. Der gesamte Zeichenvorrat umfasst die 26 Großbuchstaben von „A“bis „Z“, die 26 Kleinbuchstaben „a“bis „z“, die zehn Ziffern von „0“bis „9“und die zwei Sonderzeichen „+“und „/„. [11]

Extended Validation SSL Zertifikate

Extended Validation SSL Zertifikate (EV-SSL) sind X.509 SSL-Zertifikate, welche an strengere Vergabekriterien gebunden sind. Dies bezieht sich vor allem auf die detaillierte Überprüfung des Antragsstellers durch die CA. [20]

Favicon

Ein Favicon ist ein kleines Symbol, das unter anderem in der Adresszeile eines Browsers Links von der URL angezeigt wird. Es dient dazu der Webseite eine Wiedererkennungswert zu geben. Aktuelle Browser stellen zusätzlich sichere Verbindungen damit dar. [27]

IP-Adresse

Eine IP-Adresse ist eine Adresse in Computernetzen, wie zum Beispiel das Internet. Sie wird Geräten zugewiesen, welche an das Netz angebunden sind. Dadurch werden die Geräte erreichbar im Netz.

OSI Schichtenmodell

Es dient als Designgrundlage für Kommunikationsprotokolle.

PHP

PHP (Hypertext Preprocessor) ist eine weitverbreitete Server-Skript-Sprache, die besonders geeignet für die Webentwicklung ist. PHP Kann in HTML eingebettet werden, wird aber dann vor der Auslieferung der Webseite vom Server bearbeitet. [8]

Proxy

Proxy bedeutet Bevollmächtigter oder Stellvertreter. Ein Proxyserver ermöglicht Systemen, die keinen direkten Zugang zum Internet haben, den indirekten Zugang zum Netz.

Replay Angriffe
: Replay Angriffe sind Angriffe bei denen einmal zur Authentisierung benutzte Daten, wie z.B. verschlüsselte Passwörter, von einem Angreifer bei einem späteren Zugangsversuch wieder eingespielt werden. [6]

sniffen
: Ein Sniffer ist eine Software, die den Datenverkehr eines Netzwerkes empfangen, aufzeichnen, darstellen und gegebenfalls auswerten kann. Es handelt sich also um ein Werkzeug der Netzerkanalyse.

Acronyms

AES	Advanced Encryption Standard
CA	Certificate Authority
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
OSI	Open Systems Interconnection Reference Model
PKI	Public-Key Infrastruktur
SSL	Secure Sockets Layer
TLS	Transport Layer Security
WWW	World Wide Web

Literatur

[1] B. H. D. 2009. *SSLSTRIP*. http://www.thoughtcrime.org/software/sslstrip/index.html, 2011.

[2] K. Benton. Secure ssl tls certificate thumbprint retrieval service. *Online zugegriffen am 26.01.2012*, 2012.

[3] T. Berners-Lee. Hypertext transfer protocol – http/1.1, rfc2616. *Division of Applied Mathematics*, 1999.

[4] P. Bojanic. *The Joy of XUL*. https://developer.mozilla.org/en/The_Joy_of_XUL, 2012.

[5] I. B. Dan Boneh, Srinivas Inguva. Ssl man in the middle proxy. Technical report, Stanford Universität, http://crypto.stanford.edu/ssl-mitm/, 2007.

[6] C. Eckert. *IT-Sicherheit*. Oldenbourg Verlag, 2012.

[7] M. Ghiglieri. Thunderbird-plugin zur erkennung von anhangverdächtigen e-mails. Master's thesis, TU Darmstadt, 2009.

[8] T. P. Group. *PHP*. http://www.php.net/, 2012.

[9] P. D. D. F. W. Hesse. Client server. *Online zugegriffen am 22.01.2012*, 2011.

[10] P. D. D. F. W. Hesse. Javascript. *Online zugegriffen am 22.01.2012*, 2011.

[11] P. D. D. F. W. Hesse. Base64-code. *Online zugegriffen am 26.01.2012*, 2012.

[12] C. H. Hochstätter. Hacker stellt neue angriffe auf ssl vor. *Online zugegriffen am 26.01.2012*, 2012.

[13] lexexakt. homograph attack. *Online zugegriffen am 26.01.2012*, 2012.

[14] A. Maj. Apache 2 with ssl/tls: Step-by-step. *Online zugegriffen am 26.01.2012*, 2011.

[15] M. Marlinspike. *New Tricks For Defeating SSL In Practice*. http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf, 2011.

[16] S. D. Marsh Ray. Renegotiating tls. Technical report, Phonefactor inc., $http : //extendedsubset.com/Renegotiating_TLS.pdf$, 2009.

[17] M. Repges. Die funktionsweise von ssl. *Online zugegriffen am 26.01.2012*, 2011.

[18] K. O. Seidler. Xampp. *Online zugegriffen am 26.01.2012*, 2012.

[19] P. Seiler. Konzept und aufbau einer prototypischen public key infrastruktur auf basis von flexitrust. Master's thesis, TU Darmstadt, 2001.

[20] M. Stopczynski. Erweiterte benutzerführung für den umgang mit sicheren verbindungen in browsern. Master's thesis, TU Darmstadt, 2009.

[21] S. Systems. Wie funktioniert https? *Online zugegriffen am 26.01.2012*, 2012.

[22] A. S. Tanenbaum. *Computernetzwerke*. Pearson Studium, 2003.

[23] J. R. Thai Duong. Here come the xor ninjas. In *ekoparty Security Conference 7° edición*, $http : //www.capitancrunch.com.ar/wp - content/uploads/2011/09/ssl_jun21.pdf$, 2011.

[24] C. Veness. Movable type ltd. *Online zugegriffen am 26.01.2012*, 2012.

[25] P. D. Waidner. Fachgebiet sicherheit in der informationstechnik. *Online zugegriffen am 26.01.2012*, 2012.

[26] P. D. Waidner. Introduction to it security, SS 2011.

[27] Wikipedia. Favicon. *Online zugegriffen am 26.01.2012*, 2012.

[28] Wikipedia. Hypertext markup language. *Online zugegriffen am 26.01.2012*, 2012.

[29] J. Winkler. Http: Request-methoden. *Online zugegriffen am 26.01.2012*, 2012.