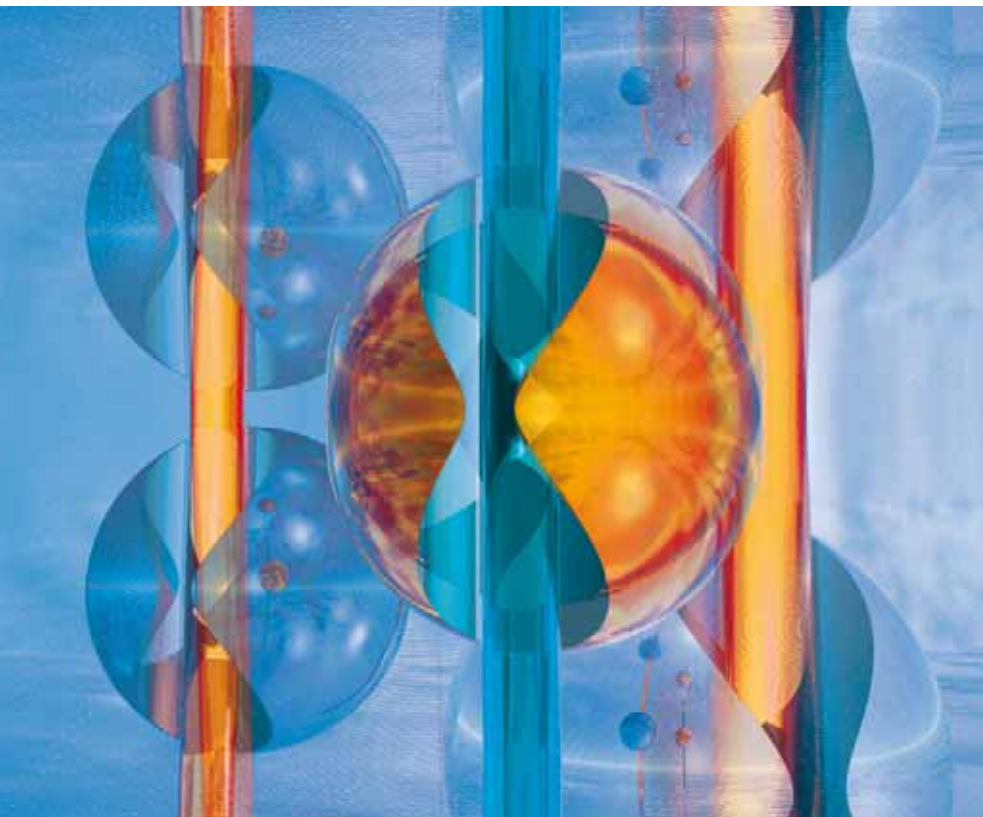


Teil 2

Authentifizierung und Session Handling



Viele Web-Anwendungen benutzen irgend eine 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. Der zweite Artikel der Reihe „Web Application Security“ behandelt Angriffstechniken wie Session-Prediction, -Interception, -Fixation oder Brute-Force-Attacken und deren Abwehr, bei der die Themen Authentifizierung und Autorisierung die Hauptrolle spielen.

Das HTTP Protokoll ist zustandslos (stateless), das heisst, dass einzelne Anfragen an den Webserver, ja selbst die eingebetteten 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 Webanwendungen die Implementierung von Sessions notwendig.

Eine Session ist eine Abfolge von zusammengehörigen HTTP-Requests eines Benutzers auf eine Applikation. Innerhalb einer Session wird ein Zustand verwaltet, wie zum Beispiel der Inhalt der 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 jedoch ein ernsthaftes Sicherheitsproblem.

Implementation von Sessions und Authentifizierung

Im Web gibt es verschiedene Varianten der Kombination von Authentifizierung und Session; für die meisten Applikationen hat sich heute jedoch ein Standardverfahren etabliert. Hierbei generiert der Webserver eine eindeutige SessionID und schickt diese in der Regel per Cookie an den Webbrowser, der dieses 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 SessionID abgefragt werden.

Die Authentifizierung eines Benutzers erfolgt über die Eingabe von Benutzernamen und Passwort auf der Webseite. Wenn diese Daten zu einem bekannten Benutzer gehören, dann ordnet der Server der bekannten SessionID den Benutzernamen als erfolgreich authentifizierten Benutzer zu. Bei allen folgenden Requests mit derselben SessionID wird dann auf Serverseite davon ausgegangen, dass es die Anfrage von dem bekannten Benutzer ausgelöst worden ist. Das bedeutet aber, dass der Zustand der Session und der Integrität der SessionID 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 der 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, zum Beispiel 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 Webapplikation sollte den Zustand idealerweise immer auf Serverseite abspeichern und an den Client nur die SessionID übertragen. Damit ist eine direkte Manipulation

mern vergeben. Auch andere selbstgebaute Verfahren sind oft nicht so sicher, wie der Programmierer ohne kryptografische Ausbildung annimmt. Eine SessionID 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 Webapplikationen ohne kryptografische Ausbildung sollte man deswegen darauf

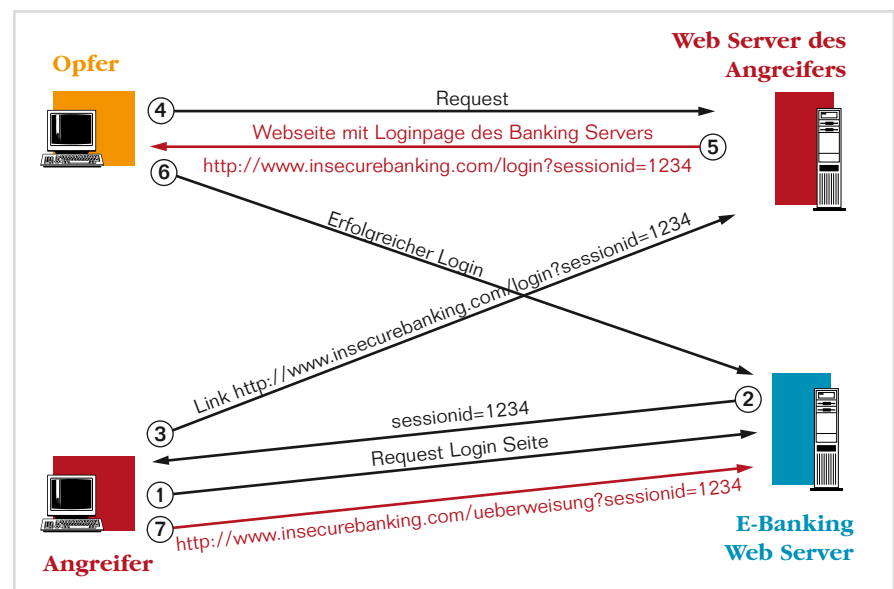


Bild: Anatomie eines Web Server-Angriffes mittels Session-Fixation.

des Zustands der Session ausgeschlossen. Die folgenden Szenarien gehen davon aus, dass auf dem Client nur die SessionID abgespeichert wird und vom Angreifer die Integrität dieser SessionID angegriffen wird.

Session ID Guessing

Der einfachste Angriff ist das Erraten einer gültigen SessionID eines anderen Benutzers. Es gibt immer noch Applikationen, die als SessionID einfach fortlaufende Num-

mern existierende Frameworks zu benutzen. Da besteht die Hoffnung, dass sich die Entwickler der Applikationsframeworks des Problems angenommen und SessionIDs richtig implementiert haben - auch wenn sich dieses Vertrauen in der Vergangenheit schon mehr als einmal als falsch erwiesen hat. Als Betreiber eines Webserver sollte man über den Einsatz von externen Web Application Security-Produkten nachdenken, die das Thema SessionID Generierung erledigen und die (eventuell unsichere) SessionID der

Applikation durch eine eigene sichere SessionID ersetzen.

Session Hijacking

Wenn ein Angreifer die SessionID 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

indirekt ausgelesen werden. Was Cross-Site-Scripting ist, wird in einem späteren Artikel genauer erläutert. Falls die SessionID 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 Webbrowser 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

Angreifer setzt hierbei irgendwann im Vorfeld die SessionID im Webbrowser des Opfers auf einen ihm bekannten Wert. Danach wartet der Angreifer darauf, dass das Opfer die Webseite benutzt und diese SessionID mit den Login-Informationen des Servers verbunden wird. (siehe Bild) Wie kann so etwas geschehen?

Wenn die Applikation die SessionID per URL und nicht per Cookie überträgt, kann der Angreifer eventuell den Benutzer überzeugen, auf einen Link mit bekannter SessionID zu klicken und dann die Applikation zu benutzen.

Auch wenn standardmässig ein Cookie benutzt wird, akzeptieren einige Webframeworks die SessionID auch in der URL. Auch wenn nur Cookies zur Übertragung der SessionID 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 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 Webapplikation wie Cross-Site-Scripting zu benutzen, um ein bestimmtes Cookie zu setzen. Was kann man als Webseitenbetreiber und Applikationsentwickler dagegen tun?

Die Webapplikation muss eine unbekannte SessionID ignorieren und stattdessen eine neue generieren. Idealerweise wird auch nach dem erfolgreichen Login eines Be-



„Eine Webapplikation sollte den Zustand idealerweise immer auf der Serverseite abspeichern und an den Client nur die SessionID übertragen. Damit ist die direkte Manipulation des Zustands der Session ausgeschlossen.“

Firmen stellen ihre Webserver bei einem Webhoster unter. Ein Angreifer kann versuchen, bei demselben Host einen Webserver zu mieten. Je nach Webhoster ist ein Mitlesen des Netzwerkverkehrs von anderen Servern möglich. Was kann man einem solchen Fall entgegenwirken?

Wenn die Sicherheit der Session irgendeine wirtschaftliche Bedeutung für den Webapplikationsbetreiber hat, dann ist der Einsatz von SSL zwingend notwendig; dadurch wird dann die SessionID nicht mehr im Klartext übertragen und ein (passives) Mitlesen des Netzwerkverkehrs bedeutet keine Gefahr mehr. Leider gibt es noch andere Varianten an die SessionID heranzukommen. Wenn die Webapplikation zum Beispiel anfällig für Cross-Site-Scripting ist, dann kann über diese Methode die SessionID

Session Hijacking kann dadurch entstehen, dass man die IP des anfragenden Webclients mit der SessionID verbindet und Anfragen mit derselben SessionID, aber einer anderen anfragenden IP-Adresse ablehnt. 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 SessionID zu erfahren, ist, dem Opfer eine bekannte SessionID unterzuschreiben. Der

nutzers eine neue SessionID generiert und die alte nicht weiter verwendet.

Session Riding

Wenn der Angreifer die SessionID 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. Das Cookie der e-Banking Applikation und damit die SessionID 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 eingeloggt Benutzer und führt die durch die Anfrage spezifizierte Aktion aus - zum Beispiel 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 Email 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 Webapplikationen zu benutzen. Auch der Webseitenbetreiber beziehungsweise der Applikationsent-

wickler 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 mit-schickt. Auch hier kann der Einsatz von spezialisierten Web Application Security-Produkten sinnvoll sein.



AlexanderMeisel
alexander.meisel
@artofdefence.com