Ajax–Risiken lassen sich eindämmen

Es gibt kaum noch eine moderne Webapplikation, die ohne Ajax auskommt. Doch mit der –Verbreitung der neuen Technik wachsen auch die potenziellen Angriffsmöglichkeiten.

Die beiden Autoren sind Security–Consultants bei Symantec.

Mit dem Ajax kann so sicher sein wie eine bisherige Anwendung, wenn man sich an die traditionellen Vorgaben für sicheres Programmieren hält.

Myspace–Wurm «Samy» wurde vielen Internetbenutzern vor Augen geführt, dass mit Ajax nicht nur Vorteile, sondern auch neue –Sicherheitsrisiken verbunden sind. Samy war der erste offiziell bekannte Wurm auf Basis von Ajax. Er pflanzte sich fort, indem er eine einfache Cross–Site–Scripting–Schwachstelle (XSS) in der für die Benutzerprofile zuständigen Funktion des Webportals ausnutzte.

Der Name Ajax steht für «Asynchronous JavaScript And XML». Schaut man sich die hinter diesen Begriffen stehenden Techniken genauer an, werden rasch die ersten Aus–wirkungen auf die Sicherheit erkennbar: Um die Asynchronität der Kommunikation zwischen Browser und Webanwendung realisieren zu können, wird das –«XMLHttpRequest»–Objekt des Browsers verwendet, das mittels Javascript (wie auch mittels anderer Scriptsprachen wie VB–Script) gesteuert werden kann. Dieses Objekt ist im Internet Explorer vor der Version 7 in Form einer zusätzlichen Aktive–X–Komponente enthalten, während es bei anderen Browsern wie Firefox und Internet Explorer 7 nativ in der Browserengine implementiert ist. Um das XMLHttpRequest–Objekt verwenden zu können, muss in den Sicherheitseinstellungen des Browsers die Ausführung von Javascript erlaubt sein. Diese Einstellung wiederum hat Auswirkungen auf die Gesamtsicherheit des Browsers, denn viele Angriffe gegen Webbrowser basieren auf der Ausführung von Scriptcode. Aus der Sicht eines Unternehmens, das eine Ajax– beziehungsweise Javascript–basierte Applikation betreibt, bleibt hier nur die Möglichkeit, den Webserver in eine vertrauenswürdige Zone des Browsers aufzunehmen. Nur so kann verhindert werden, dass beliebige Webseiten Skripte im Browser des Benutzers ausführen können.

Browser als Desktopprogramm

Unter anderem erfreut sich Ajax deshalb so grosser Beliebtheit, weil es einen Browser in die Lage versetzt, dem Anwender einen ähnlich hohen Bedienkomfort wie eine Desktopanwendung zu bieten. Realisiert wird dies hauptsächlich durch zwei Mechanismen: Zum einen ist die Ajax–Engine (dargestellt durch das XMLHttpRequest–Objekt) in der Lage, im Hintergrund kontinuierlich Daten mit dem Webserver auszutauschen, ohne dass der Benutzer eine Aktion ausführen muss – in der Regel bekommt der Anwender von dem Datenaustausch nicht einmal etwas mit. Dadurch können Inhalte der Web–seite regelmässig ohne Durchführung eines Seitenrefreshs aktualisiert werden. Zum anderen stellen moderne Webbrowser eine komplette Entwicklungsumgebung dar, in der komplexe Anwendungen unter Zuhilfenahme verschiedener Scriptingsprachen realisiert werden können. Dies führt dazu, dass bei der Entwicklung von Applikationen Teile der Anwendungslogik auf den Client ausgelagert werden. Zwar lässt sich dadurch Rechenzeit auf dem Anwendungsserver einsparen, jedoch entsteht ein erhebliches Risiko, wenn man beispielsweise bei der Entwicklung eines Online–Shops die Berechnung des Kaufpreises auf dem Client durchführt.

Asynchrone Kommunikation als CruxHier gilt eine der Faustregeln der sicheren Systementwicklung: «Benutzereingaben sind nicht vertrauenswürdig». Alles, was in dem Webbrowser verarbeitet wird, kann manipuliert werden, bevor es zum Server gesendet wird. Grundsätzlich sind solche Daten also nur auf dem Server zu bearbeiten. Lässt sich eine Auslagerung auf den Client nicht vermeiden, so ist eine serverseitige Prüfung der Daten unerlässlich.

Asynchrone Kommunikation als Crux

Durch die asynchrone Kommunikation zwischen Browser und Webapplikation ergibt sich noch ein weiterer Aspekt, der sich zu einem Sicherheitsproblem in Gestalt eines Denial–of–Service (DoS) der Webapplikation ausweiten kann. Ein Beispiel: Bei der Suche nach bestimmten Produkten in einem Online–Shop wird der Benutzer bei einer klassischen Webanwendung den Suchbegriff vollständig in ein Textfeld eingeben und danach einen Button anklicken, um die Daten vom Browser zum Server zu senden. Die Suchanfrage wird vom Server verarbeitet, das Ergebnis zum Browser zurückgesendet und von diesem dargestellt. Zur Erhöhung des Bedienkomforts würde man in einer Ajax–basierten Applikation das Textfeld um eine Auto–Vervollständigungsfunktion erweitern. Mit jedem Buchstaben, den der Benutzer eingibt, zeigt der Browser einen Vorschlag an, um die Eingabe zu vervollständigen. Hierzu sendet die Ajax–Engine im Hintergrund nach jedem eingegebenen Buchstaben eine Anfrage zum Server. Die Antwort wird dann vom Browser dargestellt.

Im Unterschied zum klassischen Modell erhöht sich allein bei diesem Beispiel die Anzahl der Anfragen und Antworten enorm. Multipliziert man das mit mehreren tausend gleichzeitig aktiven Benutzern, so kann eine Webapplikation schnell an die Grenzen ihrer Leistungsfähigkeit gelangen. Um ein solches Risiko zu vermeiden, ist ein ausführlicher Lasttest der Applikation unumgänglich – insbesondere dann, wenn eine bestehende –Applikation, für die eventuell schon ein Lasttest und ein Hardware–Sizing durchgeführt wurde, auf Ajax umgestellt wird.

Betrachtet man Ajax–Applikationen im Hinblick auf die bekannten Angriffe gegen Webapplikationen, wie beispielsweise SQL–Injection, Directory Traversal oder Cross–Site–Scripting (XSS), so zeigt sich, dass diese serverseitig genauso auftreten wie bei klassischen Webapplikationen. Gleiches gilt auch für Ajax–Anwendungen, die ihre Daten mittels XML an Web–Services übertragen. Hier kommen spezifische Angriffe, wie beispielsweise Xpath–Injection genauso zum Tragen wie bei klassischen Web–Applikationen.

Allerdings ändert sich der Angriffsvektor. Ajax–basierte Anwendungen senden die Daten in Form so genannter Sub–Requests, die mittels Javascript–Code auf dem Client erstellt werden.

Da sich aus Sicherheitssicht Ajax–Applikationen serverseitig nicht wesentlich von klassischen Webapplikationen unterscheiden, sind hier die gleichen Sicherheitsmechanismen zu implementieren. Diese bestehen insbesondere in einer strengen Filterung der Benutzereingaben sowie einer konsequenten Authentifizierung beim Zugriff auf die Serverressourcen. Denn gerade bei der massiven Verwendung von Sub–Requests mittels der Ajax–Engine besteht die Gefahr, dass für die zur Verarbeitung der Sub–Requests benötigten serverseitigen Skripte nicht die gleichen Sicherheitsmechanismen (Authentifizierung, Autorisierung und Input–Filterung) angewendet werden wie für jene Serverskripte, welche die klassischen Requests bearbeiten.

Sub–Requests können im Übrigen nur im gleichen Sicherheitskontext (Protokoll, Subdomain, Zielport)

gesendet werden wie der Rest der Webanwendung. Daher muss bereits in der Designphase einer Applikation entschieden werden, ob sensitive Daten per Ajax–Sub–Request übertragen werden sollen. Ist dies der Fall, sollte die komplette Applikation nur über Secure Socket Layer (SSL) erreichbar sein.

Angriffsziel: Client

Während sich also die serverseitigen Risiken im Vergleich zu normalen Webapplikationen kaum ändern, erreichen clientseitige Angriffe eine neue Dimension. Aufgrund der eigenständigen, im Hintergrund ablaufenden Aktivitäten der Ajax–Engine können sich clientseitige Angriffe wie beispielsweise XSS dauerhaft im Browser einklinken. Damit stellen sie ein deutlich höheres Risiko dar als bei klassischen Applikationen.

XSS–Angriffe vor Ajax hatten grundsätzlich eine sehr kurze Lebensdauer in der Applikation. Klickte ein Benutzer auf einen von einem Angreifer modifizierten Link, so wurde der darin eingebettete Script–Code einmalig ausgeführt. Wollte man dem Scriptcode jedoch eine längere Lebenszeit im Browser ermöglichen, musste man mit allerhand Tricks arbeiten – was wiederum sowohl das Risiko der Angriffserkennung durch den Benutzer als auch die Komplexität des Angriffs und somit dessen Fehleranfälligkeit erhöhte.

Mit Ajax ist es wesentlich einfacher möglich, Scriptcode so in den Browser einzubetten, dass dieser so lange im Hintergrund läuft, bis der Browser beendet wird. Damit sind beispielsweise Angriffe denkbar, die mittels einer XSS–Attacke alle beim Surfen des Benutzers anfallenden Daten im Hintergrund zum Angreifer sendet. Eine Methode, die einen solchen Angriff ermöglicht, ist das Prototype–Hijacking.

Beim Prototype–Hijacking nutzt der –Angreifer die Möglichkeit, Scriptcode zu injizieren, um bestehende Objekte zu modifizieren oder zu überschreiben. Da Javascript bei seinem objektorientierten Ansatz keine Klassen sondern Prototypen verwendet, um Objekte zu generieren, spricht man nicht von Objekt–Hijacking, sondern von Prototype–Hijacking.

Angriffe dieser Art können nur unterbunden werden, wenn seitens der Webapplikation eine strenge Filterung nach Scriptcode erfolgt. Hierbei sind natürlich auch die gängigen Encodingverfahren zu berücksichtigen, welche Angreifer gerne zum Umgehen serverseitiger Eingabefilter verwenden.

Gefährliche Frameworks und Libraries

Da die Entwicklung von Ajax–Applikationen sehr komplex werden kann, hat sich die Verwendung von Ajax–Frameworks und –Libraries durchgesetzt. Sie sollen dem Programmierer die Verwendung von Ajax erleichtern. Der Funktionsumfang dieser Frameworks reicht vom blossen Generieren des für Ajax benötigten Client–Side–Javascript–Codes bis hin zu einem Session–Management für die Ajax–Sub–Requests. Viele dieser Frameworks wurden jedoch nicht mit dem Fokus auf eine sichere Programmierung entwickelt. Daher wurden in jüngster Vergangenheit in bekannten Frameworks etliche Sicherheitsprobleme, wie beispielsweise Cross–Site–Request–Forgery (CSRF), JSON–Injections (JavaScript Object Notation), JSON–Callback–Attacks und JavaScript–Injections, entdeckt.

Damit Ajax–Anwendungen nicht automatisch die Schwachstellen der verwendeten Frameworks «erben», ist darauf zu achten, dass stets aktuelle Versionen verwendet werden, für die keine Sicherheitslücken bekannt sind. Des Weiteren empfiehlt es sich, nicht benötigte JavaScript–Code–Funktionen, die im Framework erstellt oder in einer Library enthalten sind, zu entfernen. Einem Angreifer werden so möglichst wenig Built–in–Funktionalitäten angeboten, die ihm helfen, seine Angriffe erfolgreich durchzuführen.

Ajax–Apps können sicher sein

Viele der genannten Risiken existierten in der einen oder anderen Form auch schon vor Ajax. Aus Sicht eines Sicherheitsverantwortlichen hat jedoch die Komplexität von Anwendungen insbesondere auf der Clientseite stark zugenommen. Entsprechend erhöht sich der Aufwand bei der Umsetzung bestehender Sicherheitsrichtlinien. Werden die aus der Softwareentwicklung bekannten Vorgaben der sicheren Programmierung konsequent umgesetzt, so sind Ajax–basierte Anwendungen ebenso sicher wie Anwendungen, die auf traditionellen Technologien basieren.

Praxis–Tipps

So machen Sie IhreAjax–Anwendungen sicher

• Kritische Teile der Anwendungslogik (etwa den Warenkorb) nicht clientseitig realisieren.

• Strenge Eingabefilterung aller vom Server entgegengenommenen Daten.

• Serverseitige Implementierung von Sicherheitsmassnahmen und –prüfungen; keine Auslagerung auf den Client.

• Die Anzahl der Ajax–Sub–Requests sollte so gering wie möglich gehalten werden, um die Angriffsfläche zu reduzieren.

• Session–Daten (z.B. Cookies) sollten auch bei Ajax–Sub–Requests ausgewertet werden.

• Libraries/Funktionen im Client–Side–Javascript–Code, welche direkte Cross–Domain–Requests erlauben, sollten nicht verwendet werden.

• Sofern der Zugriff auf Daten ausserhalb der applikationseigenen Domain notwendig sein sollte, wird der Einsatz serverseitiger Proxies empfohlen.

• Auf die Verwendung von JSON (Javascript Object Notation) als Format zur Datenübertragung sollte aufgrund der bestehenden Sicherheitsrisiken verzichtet werden.

• Um Cross–Site–Request–Forgery–Angriffe ver–hindern zu können, wird empfohlen, bei allen Formularen und Links eindeutige und nur einmal gültige Tokens zu übergeben.