➢ Artikel vom 8. Januar 2016

Sicherheit in Webanwendungen

Das Thema Sicherheit ist heute wichtiger als je zuvor und sollte bei einem Softwareprojekt bereits von Anfang an bedacht werden. Machen Sie es potenziellen Angreifern so schwer wie möglich. Wir zeigen Ihnen, wie.

Jedes Softwareprojekt muss sich früher oder später mit dem Thema Sicherheit beschäftigen. Die einen tun dies gewissenhafter und die anderen vernachlässigen das Thema etwas mehr. Spätestens seit Edward Snowden sollte jedoch jedem bewusst sein, dass dieses Thema gar nicht hoch genug priorisiert werden kann. Sicherlich wird es nie einen hundertprozentigen Schutz gegen alle Angriffe geben, aber es sollte einem Angreifer so schwer wie möglich gemacht werden.
Was sollte also aus der Sicht eines Softwareprojekts beachtet werden? Eine ganze Menge! Wir wollen Ihnen die wichtigsten Maßnahmen aufzeigen. Das meiste wird Ihnen hoffentlich bereits vertraut sein, aber vielleicht wird auch das ein oder andere Neue für Sie dabei sein. Keine der Maßnahmen für sich alleine reicht aus, um eine Anwendung sicher zu machen. Aber je mehr davon umgesetzt wird, umso schwerer wird es für einen Angreifer.

Authentifizierung und Autorisierung

Über die Authentifizierung soll sichergestellt werden, dass ein Benutzer des Systems auch derjenige ist, für den er sich ausgibt. In der Regel erfolgt diese über die Eingabe eines Benutzernamens und eines Passworts beim Aufruf der Startseite der Anwendung. Was ist aber, wenn ein Benutzer anstatt der Startseite eine beliebige andere Seite der Anwendung aufrufen will? Für diesen Fall muss sichergestellt werden, dass ein Authentifizierungsmechanismus existiert, der nicht umgangen werden kann. Es muss also bei dem Aufruf jeder Seite geprüft werden, ob eine gültige Authentifizierung eines Benutzers vorliegt.

Eine Authentifizierung und auch jeder andere Request, der sensible Daten

beinhalten könnte, sollte ausschließlich über einen POST-Request erfolgen. Bei einem POST-Request werden die Parameter im Rumpf des HTTP-Requests übertragen. Bei einem GET-Request sind die Parameter (zum Beispiel der Benutzername und das Passwort) ein Bestandteil der URL. Dadurch erscheinen die Parameter zum Beispiel auch in der URL-Leiste des Browsers und erscheinen somit auch in der Browser Historie. Hat ein Angreifer also Zugriff auf Ihren Browser, so kann er über die Browser Historie eventuell auch an den Benutzernamen und das Passwort gelangen. POST- und GET-Requests haben aber beide einen gemeinsamen Nachteil. Beide Request-Arten können von einem Angreifer mit einfachen Mitteln mitgelesen werden. Daher sollten die Daten grundsätzlich nur verschlüsselt übertragen werden.

Da es sich Entwickler gerne auch mal etwas leichter machen, wird häufig während der Entwicklung ein Developer-Login oder eine Backdoor eingebaut, mit dem der eigentliche Anmeldemechanismus umgangen wird. Hier sollte selbstverständlich darauf geachtet werden, dass bei einem Release sämtliche Stellen aus dem Code entfernt wurden.

Hand in Hand mit der Authentifizierung kommt die Autorisierung. Ein Autorisierungsmechanismus stellt sicher, dass jeder Benutzer nur das machen darf, für das er berechtigt ist. Auch hier gilt, genauso wie bei der Authentifizierung, dass bei jedem (sensiblen) Seiten- oder Methodenaufruf geprüft werden muss, ob der Benutzer für diese Aktion berechtigt ist. Anderenfalls kann ein Angreifer, ggf. durch Erraten einer URL, Zugriff auf einen Bereich erhalten, für den er nicht berechtigt ist.

Sessionverwaltung

Bei Webanwendungen kommt für die Kommunikation das Protokoll HTTP zum Einsatz. Da es sich hierbei um ein zustandsloses Übertragungsprotokoll handelt, wäre somit eine eindeutige Zuordnung zwischen Client und Server beim nächsten Request nicht mehr möglich. Um dennoch einen Client identifizieren zu können, erzeugt der Server beim ersten Kontakt eine eindeutige Session ID und überträgt diese an den Client. Der Client merkt sich diese Session ID und identifiziert sich bei jedem weiteren Kontakt mit dieser Session ID. So ist eine eindeutige Zuordnung möglich.

Ein Angreifer, sollte er Kenntnis von der Session ID eines Benutzers erhalten, kann

die Identität dieses Benutzers annehmen. Um sich davor zu schützen, ist bei der Erzeugung der Session ID darauf zu achten, dass die Vorhersage einer gültigen Session ID nicht möglich ist. Eine Session ID sollte daher möglichst lang und komplex sein. Ein einfaches Hochzählen der Session ID ist verständlicherweise keine adäquate Lösung. Für die Generierung wird vielmehr eine vertrauenswürdige Zufallsquelle benötigt, da ein Angreifer ansonsten durch Reverse-Engineering gültige Session IDs erzeugen könnte. Jede Session ID darf natürlich nur ein einziges Mal vergeben werden, da ansonsten bei einer doppelten Vergabe zwei Benutzer nicht mehr unterschieden werden können.

Die Session ID kann entweder in einem Cookie abgelegt oder bei jedem Request als Parameter an die URL gehängt werden. Was sind die Vor- und Nachteile? Manche Benutzer unterbinden das Anlegen von Cookies in ihrem Browser. In solch einem Fall funktioniert das Ablegen der Session ID in einem normalen Cookie natürlich nicht. Hier würde dann das Anhängen der Session ID als Parameter weiterhin funktionieren. Stellen Sie sich aber die folgende Situation vor: Ein Benutzer ist an Ihrem Onlineshop angemeldet und möchte einen Artikel einem Bekannten zeigen. Er kopiert daher die URL aus seinem Browser und schickt diese seinem Bekannten per Mail zu. Beinhaltet die URL jetzt die Session ID und klickt der Bekannte auf diese URL, hätte er unter Umständen bereits Zugriff auf das Benutzerkonto Ihres Benutzers. Beide Varianten haben also Vor- und Nachteile. Ein gemeinsamer Nachteil ist, dass beide nicht vor man-in-the-middle-Attacken schützen, bei denen ein Angreifer die Kommunikation zwischen Client und Server abhört. Aber wie bereits am Anfang des Artikels erwähnt, gibt es keinen 100%igen Schutz. Man kann es einem Angreifer nur möglichst schwer machen.

Gibt es in Ihrer Anwendung verschiedene Sicherheitsbereiche, zum Beispiel einen öffentlichen und einen nichtöffentlichen Bereich, ist es eine gute Idee, unterschiedliche Sessions IDs zu nutzen. So kann ein Angreifer, der nur Zugriff auf die öffentliche Session ID hat, keine Rückschlüsse auf die Erzeugung der nichtöffentlichen Session ID ziehen.

Eine zusätzliche Schutzmaßnahme wäre das sogenannte Session Regeneration. Bei diesem Verfahren wird die Session ID nicht dauerhaft genutzt, sondern immer wieder durch eine neue Session ID ersetzt. Erlangt ein Angreifer Kenntnis von der Session ID eines Benutzers, ist die Wahrscheinlichkeit sehr hoch, dass diese bei einem Angriff bereits wieder ungültig ist. Dieses Vorgehen schützt auch wirksam vor Session Fixation Angriffen, bei denen ein Angreifer (Kunde A) einem Benutzer

(Kunde B) eine URL mit einer gültigen Session ID unterschiebt. Kunde B klickt dann eventuell auf diese URL und loggt sich bei Ihrem Shopsystem ein. Da der Angreifer die Session ID bereits kennt, könnte er sich nun als Kunde B in Ihrem Shopsystem bewegen.

Eine Session ID sollte auch immer ein „Verfallsdatum“, einen Timeout, haben. Dies ist natürlich immer ein Drahtseilakt zwischen Usability und Sicherheit. Ist ein Benutzer in Ihrer Anwendung angemeldet, wird kurz von einem Kollegen bei der Arbeit unterbrochen, und muss sich dann gleich wieder erneut anmelden, weil die Session abgelaufen ist, führt dies natürlich schnell zu Frust. Allerdings ist es auch keine gute Idee, eine Session unbegrenzt gültig zu lassen. Hier ist immer ein Kompromiss zu suchen. Über die Logout Funktion der Anwendung muss schließlich die Session ID beim Abmelden ebenfalls ungültig gemacht werden.

Cookie Management

Neben der Session ID kann man beliebige weitere Daten in Cookies ablegen. Allerdings gibt es hier auch viele Stolpersteine, auf die geachtet werden sollte. Generell sollten Cookies keine sensiblen Daten enthalten und man sollte generell möglichst sparsam mit der Verwendung sein. Legen Sie möglichst alle Daten auf dem Server ab. In einem Shopsystem zum Beispiel die Artikelpreise im Warenkorb abzulegen ist keine gute Idee, da der Benutzer diese selbstverständlich manipulieren kann. Auch sollten dort keine Informationen abgelegt werden, die durch eine Manipulation Zugriff auf geschützte Bereiche ermöglichen. Darüber hinaus sollten der Inhalt des Cookies idealerweise verschlüsselt sein, um eine Manipulation oder Auslesen der Daten zu erschweren.

Falls es erforderlich ist, Daten in Cookies abzulegen, ist es notwendig, die Cookie-Daten vor der Verarbeitung auf dem Server zu validieren. Wie auch generell alle Daten, die vom Client zum Server geschickt werden.

Um ein Abfangen des Cookies bei der Übertragung zu erschweren, sollte serverseitig das „Secure Flag“ gesetzt werden. Über das „Secure Flag“ wird einem Browser mitgeteilt, dass er den Cookie-Inhalt nur über eine mit HTTPS gesicherte Verbindung überträgt. Dies funktioniert natürlich nur, wenn der Server auch für HTTPS konfiguriert ist.

Eine weitere sinnvolle Maßnahme ist die Absicherung eines Cookies mit den

„HttpOnly“ Flag. Mit diesem Flag kann das Auslesen des Cookies mittels JavaScript verhindert und der Angriff über eine Cross Site Scripting Lücke erschwert werden.

Eingabeprüfungen

Bei der Verarbeitung von Eingabeparametern sollte man besonders sorgfältig sein. Generell gilt, jeder vom Client an den Server übertragene Parameter muss vor der Verarbeitung vom Server(!) überprüft werden! Eine gezielte Eingabeprüfung ist ein wirksamer Schutz gegen XSS-Angriffe, SQL-Injektionen, Pufferüberläufe und andere Eingabeangriffe.

Ein erster Schritt ist die Prüfung der Parameterlänge. Gibt es zum Beispiel ein Eingabefeld für eine deutsche Postleitzahl, muss dieser Wert niemals länger als fünf Ziffern sein. Die Anwendung sollte also auf dem Server die Länge des Wertes überprüfen und gegebenenfalls die Verarbeitung abbrechen. Da eine Postleitzahl numerisch ist, kann dies ebenfalls im selben Schritt geprüft werden. Bei einem Parameter, dessen Ausprägungen bekannt sind, wäre eine Prüfung auf diese konkreten Werte am sichersten. Ein Beispiel wäre ein Feld für Artikelnummern. Die Artikelnummern des Unternehmens sind bekannt und der Wert des Feldes kann dahingehend geprüft werden. Können die Ausprägungen eines Parameters variieren, sind vielleicht andere Regeln vorhanden, die abgeprüft werden können, zum Beispiel ein bestimmter Wertebereich oder das Format einer E-Mail Adresse. Je restriktiver eine Überprüfung ist, um so sicherer. Darüber hinaus sollten auch alle Eingaben in eine kanonische Form gebracht werden.

Eingabeparameter sind nicht nur die klassischen Eingabewerte, wie man sie zum Beispiel auf einer Website hat. Immer häufiger werden auch XML Datenfeeds zur Übertragung, von zum Beispiel Produktdaten, genutzt. Ein Angreifer könnte etwa JavaScript Code in der Produktbeschreibung einschleusen. Ruft ein Kunde diesen Artikel später über die Website auf, könnte dieser eingebettete Schadcode ausgeführt werden. Auch hier ist es also wichtig, die eintreffenden Daten strikt zu überprüfen, auch wenn die Quelle des Datenfeeds vertrauenswürdig ist.

Gerne vernachlässigt werden auch Rückgabewerte von externen Prozessen. Ruft zum Beispiel ein Servlet ein externes Programm auf und verarbeitet die gegebenenfalls kompromittierte Rückantwort ohne Überprüfung, so kann auf diese Weise Schadcode in das System eingeschleust werden. Das gleiche gilt auch für Aufrufe von Web Services.

Die Verwendung von Umgebungsvariablen sollte ebenfalls möglichst auf ein Minimum reduziert werden, da deren Inhalt leicht auf einem System geändert werden kann. Verlässt sich eine Anwendung zum Beispiel auf die Umgebungsvariable PATH, wird durch eine einfache Manipulation dieser Variablen gegebenenfalls ein falscher externer Prozess geladen.

Die Erfahrung zeigt auch, dass man sich nicht auf die Inhalte von Konfigurationsdateien verlassen kann.

Wir empfehlen eine zentrale Prüfkomponente in der Anwendung einzusetzen, so dass immer klar ist, welche Prüfungen tatsächlich durchgeführt werden!

Fehlerbehandlung

Ein wichtiger, aber oft unterschätzter Aspekt sicherer Software ist die Fehlerbehandlung. Durch eine nicht ausreichende oder unvorsichtige Fehlerbehandlung kann ein Angreifer gegebenenfalls an für ihn nützliche Informationen gelangen. Generell sollte beim Auftreten einer Exception, ob erwartet oder unerwartet, immer nur eine von Ihnen kontrollierte Fehlermeldung angezeigt werden. Dem Benutzer dürfen keine Systemfehler angezeigt werden. So wird verhindert, dass der Angreifer zum Beispiel Kenntnis von der verwendeten Datenbanktechnologie oder anderen internen Informationen erhält. Wir empfehlen daher immer eine gut durchdachte und zentrale Fehlerbehandlung in Ihrem Softwareprojekt. Neben der Erhöhung der Sicherheit hat eine zentrale Fehlerbehandlung noch weitere Vorteile: Der Wartungsaufwand bei der Entwicklung wird gesenkt, im Fehlerfall bleibt die Anwendung in einem sicheren und stabilen Zustand und Ressourcen werden kontrolliert freigegeben.

Zum Thema Fehlerbehandlung gehört es auch, das Auftreten von Ausnahmesituationen im Vorfeld zu verhindern, wo es möglich ist. So sollten Rückgabewerte von Methodenaufrufen stets geprüft werden und ungültige Zustände abgefangen werden.

Logging

Aber auch das Logging einer Anwendung ist wichtig für die Sicherheit, da ohne Logging ein Angriff oft nur schwer erkannt werden kann. Auf was sollte also beim Logging geachtet werden? Generell sollten in einem produktiven System keine

sensiblen Daten geloggt werden. Die Ausgabe von SQLs, Passwörtern und anderen sensiblen Daten, die während der Entwicklung eventuell nützlich sind, sollten unterbleiben. Stellen Sie also sicher, dass mögliche Debug-Einträge aus der Entwicklung nicht mitgeloggt werden.

Was aber auf jeden Fall mitgeloggt werden sollte, sind erfolgreiche und auch fehlgeschlagene Authentifizierungsversuche an der Anwendung. Dies erleichtert die Erkennung von zum Beispiel Brute Force Attacken ungemein. Interessant sind auch Validierungsfehler von Eingabewerten, falsche Parameternamen, Client-Operationen wie CREATE, UPDATE oder DELETE, sowie mögliche Fehler beim Session Management. Aber auch hier ist darauf zu achten, dass keine sensiblen Daten mitgeloggt werden.

Darüber hinaus sollten natürlich auch Applikationsfehler geloggt werden. Dies ist nicht nur für die Fehlersuche sehr nützlich, sondern macht gegebenenfalls ersichtlich, ob Fehlermeldungen gehäuft auftreten, was ebenfalls ein Zeichen für einen Angriffsversuch sein kann.

Fazit

Wie bereits gesagt, sorgt keine dieser Maßnahmen allein für eine sichere Anwendung bzw. wird es je eine 100%ige Sicherheit geben. Aber jede dieser Maßnahmen macht es einem Angreifer ein Stück schwerer und verschafft Ihnen eventuell die Zeit, um einen Angriff rechtzeitig zu erkennen und Gegenmaßnahmen zu ergreifen.

Ansprechpartner:

Michael Linke
Informatiker

Beckmann & Partner CONSULT

Telefon: 0521 2997320
m.linke@beckmann-partner.de
www.beckmann-partner.de