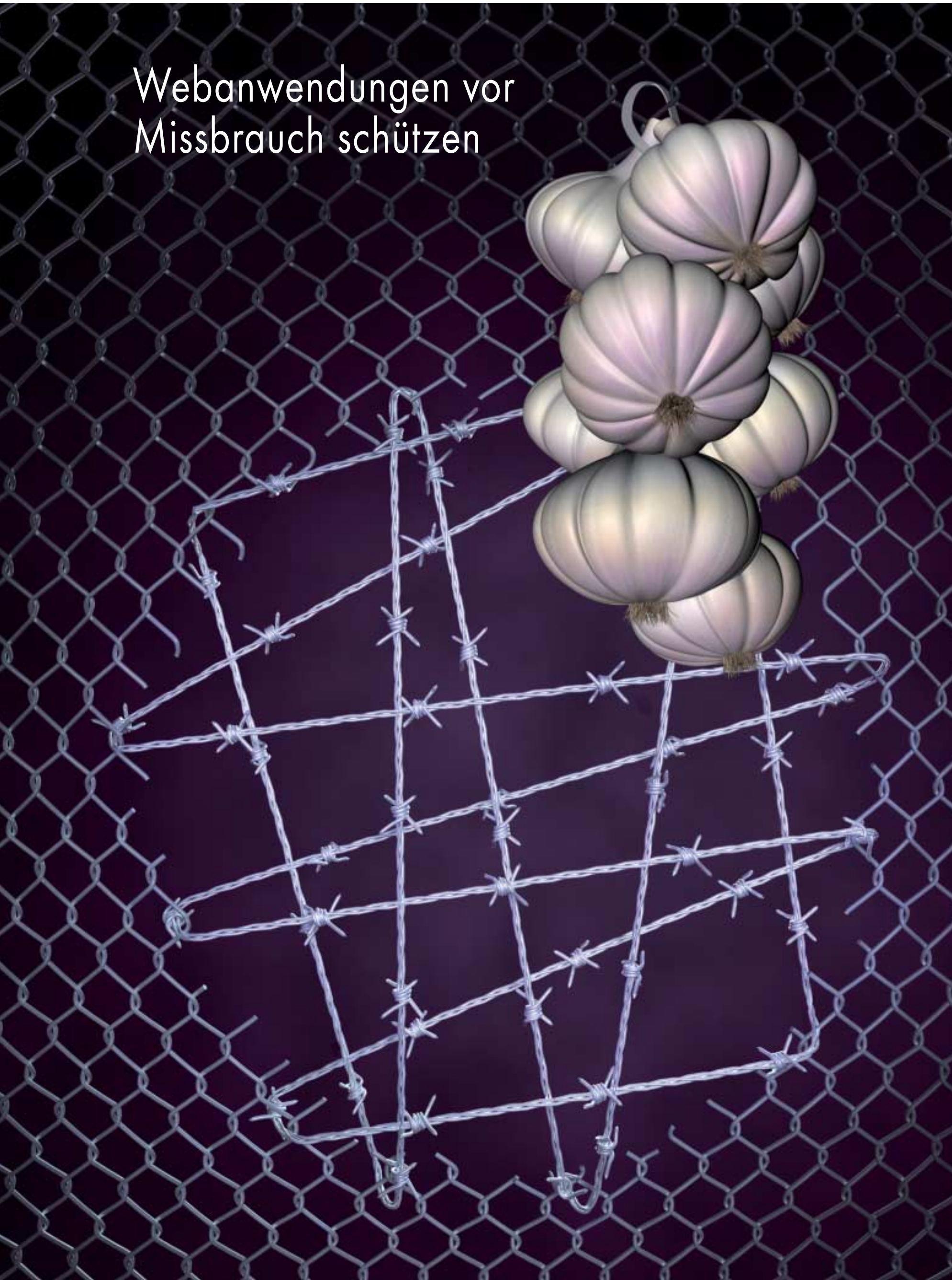


Abgedichtet

René Keller, Jörn Wagner, Martin Wundram

Dass Webanwendungen extrem angreifbar und beliebte Ziele von Hackern sind, hat sich in den letzten Jahren herumgesprochen. Doch nicht alle Webseitenbetreiber berücksichtigen, dass mehr Sicherheit nicht mit Einzelmaßnahmen, sondern nur mit einem runden Konzept zu erreichen ist.

Webanwendungen kranken oft gleich an mehreren Stellen: XSS-Schwachstellen, SQL-Injections, Information Disclosure und vieles mehr. Selbst zunächst harmlos wirkende Probleme werden von Angreifern mit fatalen Folgen ausgenutzt. Dagegen hilft nur die Berücksichtigung von Sicherheitsaspekten im gesamten Entwicklungsprozess.

Im Folgenden ist anhand einiger aktueller Beispiele zu sehen, welche Gefahren in den einzelnen Phasen des Entwicklungsprozesses einer Webanwendung von der Konzeption über die Entwicklung bis zum Betrieb lauern und wie sie sich beheben lassen.

Besondere Bedeutung kommt der Phase der Konzeption einer Webanwendung zu. In ihr können die Verantwortlichen neben der eigentlichen Anwendung die notwendigen Sicherheitsmaßnahmen planen und potenzielle Bedrohungen oder Auswirkungen von Fehlfunktionen bereits zu Beginn berücksichtigen. Die Sicherheitsarchitektur muss Teil der Anwendungsarchitektur sein. Jedem an Entwicklung und Betrieb einer Webanwendung Beteiligten muss klar sein, welcher Sicherheitsbedarf in welchem Teilbereich festgelegt ist und dass nur eine vorab geplante, stringente und einheitliche Arbeit zum erforderlichen Gesamtmaß an Sicherheit führen kann. Verschiedene Probleme lassen sich so schon während der Konzeption identifizieren und lösen.

Kapselung der Komponenten

Ein grundlegendes Merkmal sicherer Webapplikationen ist die Architektur der Kapselung aller Komponenten. Dahinter steckt, dass man bei der Konzeption der Software nicht nur den Eingaben des Webseitenbenutzers misstraut, sondern jeder Komponente, die Daten an eine andere weiterleitet. Jede Komponente muss übergebene Daten eigenständig auf Gültigkeit validieren.

Als Beispiel sei ein für Mail-Header-Injection anfälliges Kontaktformular genannt, das aus den Benutzerdaten eine E-Mail erstellt und einer E-Mail-Komponente übergibt. Schon wenn einer der beiden Verarbeitungsteile wirksame Maßnahmen gegen Header-Injection umsetzt, wird ein Angriff keinen Erfolg haben. Die Praxis zeigt, dass manche Entwickler sich mit der Absicherung einer Komponente zufriedengeben. Das kann jedoch noch keine verlässliche

© Copyright by Heise Zeitschriften Verlag GmbH & Co. KG. Veröffentlichung und Vervielfältigung nur mit Genehmigung des Heise Zeitschriften Verlags.

Lösung sein, da einzelne Komponenten gelegentlich auch gegen andere ausgetauscht werden.

Ein weiteres Beispiel liefert ein bekannter Provider für Internettelefonie. Das Aufladen des Kontos funktioniert über ein Auswahlformular, das nur Beträge ab 10 EUR aufwärts anbietet. Der Server prüft diese Mindestgrenze jedoch nicht. Der Datenstrom kann also problemlos so manipuliert werden, dass ein Kunde auch geringere Beträge aufladen kann. Da für den Provider für jede durchgeführte Lastschrift Kosten entstehen, ist nachvollziehbar, dass beispielsweise 1000 manipulierte Buchungen zu je einem Cent mehr Kosten als Nutzen bringen. Eine separate Absicherung in jeder Komponente würde solche Manipulationen verhindern.

Wie eine sichere Architektur aussieht

In einer sicheren Architektur kommen den einzelnen Komponenten folgende Aufgaben zu:

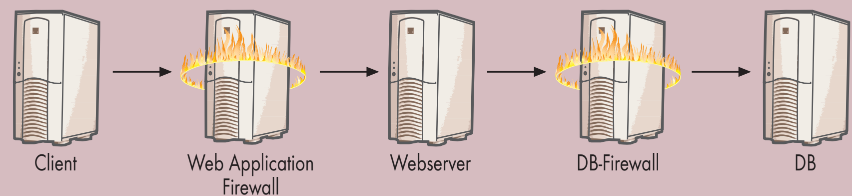
Client: Dem (möglicherweise feindlichen) Client ist aus Sicht der Webanwendung grundsätzlich zu misstrauen, clientseitige Prüfungen von Eingaben dienen dem Schutz vor versehentlich falschen Eingaben.

Web Application Firewall: Sie prüft, ob die Nutzung der Webanwendung zulässig ist, zum Beispiel über die Anzahl der Zugriffe. Außerdem soll sie Angriffe wie Denial of Service oder Cross-Site Scripting abfangen.

Server: Der Webserver überprüft alle Eingabedaten. Auf ihm lassen sich durch entsprechende Konfiguration oder Programmierung Maßnahmen gegen Session Fixation (siehe S. 48), Cross-Site Request Forgery (CSRF) und andere Angriffe ergreifen.

Datenbank-Firewall: Diese Firewall soll Datenbankabfragen kontrollieren und Standardangriffe abfangen.

Datenbank: Datenbankseitig trägt ein Berechtigungskonzept zur Sicherheit bei. Bei sicherheitskritischen Daten sollte man mit sogenannten „Stored Procedures" arbeiten (siehe Listings).



Ein gewisses Maß an Sicherheit bringt es, jede Komponente der Anwendung zu kapseln. Das heißt, jedes System prüft unabhängig von den anderen die übergebenen Eingabewerte.

Security by Obscurity

Wie beim Militär legen auch in der Geschäftswelt die Verantwortlichen in der Regel großen Wert auf die Geheimhaltung vertraulicher Informationen – etwa sensible Daten, Geschäftslogik und deren Realisierung in Programmcode. Verschleierung darf allerdings nicht die einzige Sicherheitsmaßnahme sein. „Security by Obscurity" meint in diesem Fall, dass die Information lediglich an einem nichtgenannten Speicherort versteckt wird, ohne weitere Sicherheitsmechanismen wie Passwortschutz et cetera.

Ein beliebtes Beispiel sind nichtverlinkte und damit vermeintlich geheime Unterverzeichnisse der Webanwendung (z. B. TempData), in denen sie ihre temporäre Daten ablegt. Durch Ausprobieren oder Insider-Informationen von böswilligen Entwicklern können Angreifer ohne große Mühe Daten ausspähen. Weiterhin überträgt eine Webanwendung oftmals Informationen in Hidden-Feldern von Formularen „versteckt", wie die Empfängeradresse bei einem Kontaktformular. Auch hier sollten die Verantwortlichen in der Konzeptionsphase den Sicherheitsbedarf aller Daten und Prozesse festlegen.

Missbräuchliche Massennutzung

Aspekte der Nutzungsfrequenz angebotener Dienste sind keine Schwachstellen im engeren Sinne und werden daher leicht bei der Realisierung von Webanwendungen übersehen. Ein Beispiel ist eine ungesicherte „Passwort vergessen"-Funktion eines Telekommunikationsanbieters, die an die eingegebene Handynummer ein neues Passwort verschickt oder mitteilt, dass die Nummer nicht registriert ist.

Daraus ergeben sich gleich mehrere Angriffsszenarien: Ein Täter kann massenhaft Anfragen an diesen Dienst senden und so verhältnismäßig leicht herausfinden, welche Nummern bei diesem Anbieter registriert sind. Ferner kann er über diesen Dienst eine Flut von SMS-Nachrichten an ein Opfer auslösen, das daraufhin seinen Mobilanschluss praktisch nicht mehr verwenden kann. Das Opfer könnte hier den Dienstleister in Regress nehmen, da die Webanwendung nicht ausreichend gesichert ist. Und schließlich entstehen dem Dienstleister pro gesendeter SMS Kosten. Mit einem einfachen Skript kann ein Angreifer einen beachtlichen Schaden bei dem Unternehmen verursachen.

- Nicht oder wenig angreifbare Webanwendungen sind nur zu realisieren, wenn man den Sicherheitsfokus über den gesamten Entwicklungsprozess von der Konzeption bis hin zum Betrieb beibehält.
- Gegenmaßnahmen ergreifen und Hilfswerkzeuge wie Webapplikations- oder Datenbank-Firewalls konfigurieren kann nur, wer die potenziellen Angriffe auf Webanwendungen bis ins Detail kennt.
- Nicht vergessen sollte man, dass Angriffe auf Webanwendungen auch von innen kommen können. Passworttausch und ähnliche unsichere Gebräuche im Unternehmen sollten der Vergangenheit angehören.

Unterstützung durch Log-Management

Die Frage der Haftung im Missbrauchsfall kann zwischen Betreiber und (externem) Entwickler zu Auseinandersetzungen und zu Regressforderungen an den Entwickler führen. Dieser sollte daher früh planen, in welchem Umfang angebotene Dienste genutzt werden sollen und dürfen. Es bietet sich an, dies mit der Kapazitätsplanung zu kombinieren.

Den Wert von Log-Dateien unterschätzen viele und vernachlässigen de-

© Copyright by Heise Zeitschriften Verlag GmbH & Co. KG. Veröffentlichung und Vervielfältigung nur mit Genehmigung des Heise Zeitschriften Verlags.

ren Management. In vielen Anwendungsfeldern, etwa beim Betrieb eines Onlineshops, ist das Protokollieren von Geschäftsvorfällen unerlässlich. Der Entwickler muss genau prüfen und festlegen, für welche Empfänger welche Informationen über welche Protokollwege zu speichern sind. Welcher Kunde welches Produkt von welcher IP-Adresse aus gekauft hat, ist für den Betreiber eine sicherlich aussagekräftige Information. Jedoch sollte der Zuständige bei der Konzeption einer Webseite darauf achten, nur relevante Informationen in einer Log-Datei zu speichern. Das ist datenschutzkonform („Datensparsamkeit und Datenvermeidung“ laut Bundesdatenschutzgesetz) und hält den Schaden bei missbräuchlichem Zugriff gering.

Mit Verschlüsselung und beschränktem Zugriff

Sobald Systeme personen- oder benutzerabhängige Daten wie Bank- oder Kreditkartendaten, Passwörter und Adressen protokollieren, sollten die Protokolldateien vollständig verschlüsselt sein. Ferner muss der Kreis der Personen und Systeme, der Zugriff auf diese Daten hat, klar definiert werden. Für den Entwickler einer Webanwendung ist es beispielsweise unerlässlich, einen Einblick ins Fehlerlog des Webservers oder einer Protokollschicht der Webanwendung zu erhalten. Aber muss er auch nachvollziehen können, welcher Kunde mit welcher Kreditkarte einen Kauf getätigt hat? Die Konzeption soll eine klare Definition liefern, welche Protokoll- und Fehlernachrichten der Benutzer der Anwendung, der Entwickler, der Betreiber oder weitere Personen erhalten. Personenbezogene Protokolldaten sollten von der Generierung bis zur Archivierung verschlüsselt gespeichert werden und nur einem definierten Kreis zugänglich sein.

Die Entwicklungsphase setzt die Planungsergebnisse der Konzeption um und ist damit ebenso wichtig in Bezug auf die Sicherheit bei der Realisierung von Webanwendungen. Hier gilt es, die Sicherheitsanforderungen aus der Konzeption von der Softwareentwicklung sauber zu übernehmen und gleichzeitig die Voraussetzungen für einen sicheren Betrieb zu schaffen. Ein feststehender Release-Termin darf nicht dazu führen, dass Sicherheitsanforderungen lasch umgesetzt werden oder gar völlig wegfallen.

Oftmals gerät ein Angreifer an sicherheitskritische Informationen, die die Struktur der Webanwendung selbst betreffen. Systeminterne Details wie Fehlermeldungen, Kommentare oder Verzeichnisinhalte einer unsicher programmierten Anwendung lassen sich häufig leicht erspähen und für gezielte Nachforschungen missbrauchen. Der Fachbegriff für die unfreiwillig offengelegten Daten lautet „Information Disclosure“.

Ein Beispiel dafür, das auf den ersten Blick harmlos wirkt, liefern die Betreiber einer mehrsprachig verfügbaren Webdatenbank, auf der Kunden persönliche Gegenstände registrieren können. Die Betreiber suggerieren auf der Homepage durch Statistiken einen vermeintlich großen Datenbestand und eine hohe Nachfrage nach den registrierten Gegenständen. Die Suchmaske der Webseite offenbart jedoch nach der Eingabe des Suchstrings „^“ ihren gesamten Datenbestand: gerade einmal 200 Einträge. Da die Ergebnisliste einen Link zu jedem Artikel und dessen Details, unter anderem das Registrierungsdatum, enthält, lässt sich so äußerst einfach ein Diagramm des Webseiten-„Erfolgs“ erstellen.

Auf den Geschäftserfolg eines Unternehmens kann man auch anhand anderer Informationen auf der Webseite schließen – eine vielfach unterschätzte Tatsache. Fortlaufende Produkt- oder Kundennummern offenbaren Interessierten wertvolle Informationen, wenn sie in der Webanwendung etwa über *GET*-Parameter oder „versteckt“ in Formularen öffentlich gemacht werden.

Geschwätzige Suchformulare

Wenn Webserverbetreiber bei der Produktsuche durch Platzhalter oder leere Suchbegriffe das Anzeigen aller Produkte erlauben, lässt sich das auch nicht mit einer Begrenzung der angezeigten Treffer pro Seite verhindern, wenn der Benutzer diese leicht manipulieren kann. Bei einem solchen, in der Praxis leider häufig anzutreffenden Suchformular kann sich die Konkurrenz stets per Klick auf dem Laufenden halten, wie viele Produkte im gesamten Sortiment und in einzelnen Kategorien verfügbar sind.

Um derartige Informationslecks zu vermeiden, muss der Verantwortliche klar definieren, welche Daten die Webanwendung entgegennehmen und welche sie ausgeben darf. Der gängigste Ansatz ist hier auf der Eingabeseite die Definition gültiger Eingabewerte. Dabei müssen Entwickler nicht nur das syntaktische Format der übergebenen Daten im Blick haben (etwa die Prüfung auf Ziffern), sondern sie außerdem semantisch auf Sinngehalt überprüfen, zum Beispiel, indem sie einen Wertebereich definieren.

Auf der Ausgabeseite empfiehlt es sich, möglichst wenig technische Informationen preiszugeben. Zum einen sollten Entwickler die Ausgabe detaillierter Fehlernachrichten unterdrücken. Zum anderen sollten sie sich die Frage stellen, ob die jeweilige systeminterne Information – etwa der künstliche Primärschlüssel eines Benutzers – wirklich herausgegeben werden muss. Vielfach lässt sich die Herausgabe der Daten durch ein intelligentes Sitzungsmanagement („Session Handling“) ohne Beeinträchtigung der Funktion vermeiden.

Dazu speichert man die Daten auf dem Server und übermittelt dem Kun-

Listing 1: Stored Procedure "Login"

```
# @param string Name Der eingegebene Benutzername vom Login-Formular
# @param string Pw Das eingegebene Passwort vom Login-Formular
# @return integer Ergebnis Das Ergebnis der Login-Prozedur: 1 = Login OK, -1 = Benutzer gesperrt, -2 = Benutzer gelöscht
delimiter //
CREATE PROCEDURE `Login`(Name VARCHAR(255), Pw VARCHAR(255))
READS SQL DATA
BEGIN
  DECLARE Ergebnis TINYINT;
  SELECT COUNT(*) INTO Ergebnis FROM Benutzer WHERE Benutzername = Name;
  IF Ergebnis > 0 THEN
    SELECT -1 INTO Ergebnis FROM Benutzer WHERE Benutzername = Name AND AnzahlFehlLogins >= 10 LIMIT 1;
    SELECT -2 INTO Ergebnis FROM Benutzer WHERE Benutzername = Name AND Geloescht = 1 LIMIT 1;
  END IF;
  IF Ergebnis = 1 THEN
    SELECT COUNT(*) INTO Ergebnis FROM Benutzer WHERE Benutzername = Name AND Passwort = SHA1(CONCAT(Hash, Pw)) LIMIT 1;
    IF Ergebnis = 1 THEN
      UPDATE Benutzer SET AnzahlFehlLogins = 0, LetzterLogin = NOW() WHERE Benutzername = Name LIMIT 1;
    ELSE
      UPDATE Benutzer SET AnzahlFehlLogins = AnzahlFehlLogins + 1 WHERE Benutzername = Name LIMIT 1;
    END IF;
  END IF;
  SELECT Ergebnis;
END;
//
```


© Copyright by Heise Zeitschriften Verlag GmbH & Co. KG. Veröffentlichung und Vervielfältigung nur mit Genehmigung des Heise Zeitschriften Verlags.

Listing 2: Stored Procedure "Benutzerdaten"

```
delimiter //
CREATE PROCEDURE Benutzerdaten(Benutzertoken VARCHAR(40))
READS SQL DATA
BEGIN
  # !!! nicht ausgegeben werden: Benutzer-ID, Passwort, Hash und Token !!!
  SELECT Benutzername, Mail, Vorname, Nachname, AnzahlFehlLogins, Gesperrt, Geloescht FROM Benutzer WHERE Token =
  Benutzertoken LIMIT 1;
END;
//
```

Listing 3: Stored Procedure "PasswortZuruecksetzen"

```
delimiter //
CREATE PROCEDURE PasswortZuruecksetzen(Benutzertoken INT, PasswortNeu VARCHAR(255), Zufallstoken VARCHAR(100))
READS SQL DATA
BEGIN
  DECLARE Ergebnis TINYINT;
  SELECT COUNT(*) INTO Ergebnis FROM Benutzer WHERE ID = Benutzertoken;
  IF Ergebnis = 1 THEN
    UPDATE Benutzer SET Passwort = SHA1(CONCAT(Zufallstoken, PasswortNeu)), Hash = Zufallstoken, AnzahlFehlLogins = 0
    WHERE ID = Benutzertoken LIMIT 1;
  END IF;
END;
//
```

den nur ein Cookie mit einer Sitzungsnummer. Sollte eine Preisgabe dennoch erforderlich sein, bietet eine symmetrische Verschlüsselung die Möglichkeiten, sie ohne Offenlegung von der Webanwendung zum Browser und wieder zurück zu übertragen beziehungsweise Steuerinformationen sicher in Cookies im Browser des Benutzers zu hinterlegen.

Schwachstelle Datenbank

Obwohl SQL-Injections (siehe S. 40) in der Fachwelt seit Jahren Thema sind, finden sich auf Mailinglisten wie Full Disclosure oder Bugtraq regelmäßig neue Angriffstüren für solche Angriffe in Softwarelösungen. Dabei sind SQL-Injections verhältnismäßig einfach zu verhindern. Eine klassische Gegenmaßnahme ist der Einsatz eines Datenbankabstraktionslayers (DBAL) und die Überprüfung aller übertragenen Daten auf Gültigkeit.

Mittels eines DBAL können Datenbankabfragen über eine definierte Schnittstelle erfolgen. Neben der Vorgabe, dass der Anwender bestimmte Variablen nur mit bestimmten Typen füllen darf, bieten viele DBALs eine integrierte Erkennung von SQL-Injections. Kommt die Verwendung von DBAL nicht infrage, muss der Entwickler selbstständig die Gültigkeit der übertragenen Daten prüfen und gegebenenfalls sicherstellen, dass bestimmte Steuerzeichen wie Hochkommata, Anführungszeichen, Semikola oder Bindestriche maskiert werden.

Weniger thematisiert ist die Verwendung von Stored Procedures. Mit ihnen kann der Entwickler im Datenbankmanagementsystem (DBMS) Verarbeitungsabfolgen hinterlegen, die über spezielle Datenbankabfragen aufgerufen werden können. Seit Version 5.1 steht diese Funktion im weitverbreiteten DBMS MySQL zur Verfügung. Der Einsatz von Stored Procedures kann den Zugriff auf sicherheitskritische Daten über das herkömmliche Berechtigungs-

Anzeige

© Copyright by Heise Zeitschriften Verlag GmbH & Co. KG. Veröffentlichung und Vervielfältigung nur mit Genehmigung des Heise Zeitschriften Verlags.

konzept hinaus regulieren. Beispiel sei hier eine Webanwendung, auf der sich Benutzer registrieren und ihr Profil bearbeiten können. Die Webanwendung muss somit über lesende (*SELECT*) und schreibende (*INSERT*) Berechtigungen verfügen. Ist es einem Angreifer, etwa durch Ausnutzung einer Sicherheitslücke, möglich, Quelltext in der Webanwendung auszuführen, hat er leichtes Spiel und kann mit einer Datenbankabfrage alle Benutzerdaten auslesen.

Stored Procedures regeln Zugriff

Mit Stored Procedures kann man die Verwaltung der Benutzerdaten realisieren, ohne dass der Datenbankbenutzer Zugriff auf die Tabelle der Benutzerdaten erhält – weder lesend noch schreibend, die Tabelle existiert für den Benutzer überhaupt nicht. Der Zugriff erfolgt komplett über Stored Procedures. Sämtliche Aktionen, von der Benutzerregistrierung über Authentifizierung bis zum Zurücksetzen von Passwörtern, finden in Stored Procedures statt. Die Quelltexte sind für die Webanwendung verborgen im Datenbankmanagementsystem hinterlegt, der Datenbankbenutzer hat lediglich die Berechtigung, die Prozeduren aufzurufen.

Die Listings zeigen beispielhaft einige Basisfunktionen für die Benutzerverwaltung, umgesetzt mit MySQL und Stored Procedures. Die Prozedur „Login“ (Listing 1) überprüft die Gültigkeit der übergebenen Kombination aus Benutzername und Passwort. Ist die Kombination korrekt, liefert die Funktion 1 zurück, bei falscher Kombination -1. Nach zehn erfolglosen Versuchen wird -2 zurückgegeben und der Benutzer ist gesperrt.

Diskretes Auslesen

Die Prozedur „Benutzerdaten“ (Listing 2) liest die Daten eines konkreten Benutzers aus, ohne das Passwort und andere sicherheitskritische Daten preiszugeben. Das Auslesen aller Benutzerdaten ist selbst unter Kenntnis der Datenbankzugangsdaten der Webanwendung nur durch Erraten von Benutzernamen möglich, da die Funktion höchstens einen Datensatz zurückgibt. Hat ein Benutzer sein Passwort vergessen, kann es über die Methode „PasswortZuruecksetzen“ (Listing 3) erneuert werden.

Cross-Site Request Forgery (CSRF) ist ein Angriff, bei dem jemand unberechtigt Daten in einer Webanwendung verändert. Hierzu missbraucht er einen Benutzer einer Webanwendung, der mit ausreichenden Rechten für den Angriff ausgestattet ist. Mit technischen Maßnahmen oder Überredungskunst („Social Engineering“) bringt er den Benutzer dazu, einen kompromittierten HTTP-Request (etwa einen einfachen Link, der oft hinter einer Kurz-URL versteckt ist) auszuführen und somit die vom Angreifer gewünschte Funktion auszuführen – etwa das Löschen von Benutzerdatensätzen beim Aufruf von http://example.com/user/delete/id/1…

CSRF lässt sich insbesondere durch folgende Maßnahmen wirksam unterbinden: Zunächst sollten Entwickler grundsätzlich bei allen Formularen *POST*-Requests erzwingen und auch nur per *POST* übergebene Variablen auswerten. Darüber hinaus empfiehlt es sich, zufällig generierte Token in jedes Formular zu integrieren, um sicherzustellen, dass auch wirklich der Benutzer das Formular für eine bestimmte Aktion aufgerufen hat. Besonders kritische Anwendungen sollten zudem eine Nachfrage („Soll der Datensatz wirklich gelöscht werden?!“) starten oder eine nochmalige Eingabe des Benutzerpassworts erzwingen.

Auch bei der Session Fixation missbraucht der Angreifer einen legitimen Benutzer, er übernimmt dessen Sitzung. Wie der Angriff funktioniert und was man dagegen unternehmen kann, beschreibt der Artikel „Geklaute Sitzung“ auf Seite 48.

Wartung nach Freischaltung

Bei der Inbetriebnahme einer Anwendung ist besondere Sorgfalt gefragt. Denn wenn die Anwendung erst einmal öffentlich zugänglich ist und Sicherheitslücken offenstehen, ist es nur eine Frage der Zeit, bis sie auch ausgenutzt werden – vom Schaden abgesehen, leidet oft noch der Ruf des Unternehmens.

Secure Deployment ist der Oberbegriff für eine Reihe von Best Practices, die zum Ziel haben, eine unter den beschriebenen Aspekten erstellte sichere Anwendung auch sicher zu installieren, zu betreiben und zu warten. Ziel ist es, alle erdenklichen Wege abzusichern, auf denen ein Angreifer unberechtigterweise Daten abgreifen oder sich einen Zugang verschaffen kann. Voraussetzung ist eine entsprechende Konzeption und Entwicklung der Anwendung unter Sicherheitsaspekten. Selbst das beste Secure Deployment kann nicht verhindern, dass Fehler in der Sicherheitsarchitektur ausgenutzt werden.

Der wohl beliebteste sicherheitskritische Fehler bei der Inbetriebnahme einer Anwendung ist, dass bei der Entwicklung notwendige und hilfreiche Debug-Meldungen und Zusatzinformationen (siehe „Information Disclosure“) für jeden Nutzer über Wege abrufbar sind, die eigentlich den Entwicklern und Programmierern vorbehalten sein sollten. Hier hilft der MVC-Ansatz (Model-View-Controller, zu deutsch Modell-Präsentation-Steuerung) beim Trennen der Auskunftsfreudigkeit der Anwendung je nach Zielgruppe.

Zielgruppenspezifische Fehlermeldungen

Unterschiedliche Controller für die Entwicklung und den Livebetrieb helfen, kontextbezogene Sicherheitseinstellungen vorzunehmen. So kann beispielsweise eine Fehlermeldung für den normalen Benutzer lauten „Es ist ein Datenbankfehler aufgetreten“. Fatal wäre, wenn in einer produktiven Anwendung die Meldung „Fremdschlüsselbeziehung zwischen Tabelle ‚Benutzer‘ und ‚Gruppe‘ über BenutzerID fehlt“ erscheinen und damit die interne Struktur der Datenbank offengelegt würde, was wiederum SQL-Angriffe begünstigte.

Diese Information ist aber für einen Entwickler auf Fehlersuche unersetzlich, weshalb er sie durchaus im Produktivsystem abrufen können muss. Daher sollte der Controller für die Entwickleransicht entsprechend geschützt sein mit einer Kombination aus asymmetrischen Schlüsselpaaren („Private/Public Key“), Passwortschutz und/oder IP-Sperre. Zu oft vertrauen Entwickler darauf, dass schon niemand errät, wo der „FrontendDevelopmentController(.php)“ liegt (Stichwort „Security by Obscurity“).

Einfache Problemlösung

Trivial zu vermeiden, aber immer noch von Zeit zu Zeit anzutreffen, ist eine falsche Konfiguration des Webservers. Wenn dieser freimütig alle Dateien ausliefert, die der Benutzer anfragt, kann ein Angreifer auch an Konfigurationsdateien mit der Endung .conf oder .inc gelangen und so interne Informationen

© Copyright by Heise Zeitschriften Verlag GmbH & Co. KG. Veröffentlichung und Vervielfältigung nur mit Genehmigung des Heise Zeitschriften Verlags.

wie Passwörter erlangen oder Rückschlüsse auf die Struktur der Anwendung ziehen.

Um generelle Angriffsmöglichkeiten zu minimieren, bietet sich sich der Einsatz sogenannter Web Application Firewalls (WAF) an. Für den beliebten Apache Webserver gibt es etwa das Modul „mod_security“. Einmaliges Installieren und regelmäßige Updates wie etwa bei einem Virenscanner reichen hier aber nicht. Obwohl einige Produkte umfassende Regelwerke von Haus aus mitbringen, muss in jedem Fall eine Anpassung an die Logik und Struktur der konkreten Anwendung erfolgen.

Hilfsmittel Applikations-Firewall

Eine WAF verhindert unberechtigte Zugriffe auf die Anwendung, indem sie anhand eines Regelwerks jede Anfrage beurteilt und sie entweder als zulässig einstuft, eine Warnung ausgibt oder den Zugriff verhindert. Sie lässt nur gültige oder wenig bedenkliche Anfragen zu und blockiert alle anderen prophylaktisch. Dadurch ist der Grad der Sicherheit nicht vom Erfahrungshorizont und der Vorstellungskraft des Administrators abhängig.

Die Kombination beider Ansätze bietet umfassenden Schutz, sofern sie richtig angewendet werden. Damit das „Whitelist“-Prinzip nicht zur Einschränkung des Funktionsumfangs der Anwendung führt, muss man eine WAF sorgfältig konfigurieren. Es hat sich in der Praxis bewährt, zunächst mit einem restriktiven Regelwerk anzufangen und beim Testen der Anwendung nach und nach die Einschränkungen zu lockern.

In dieser Trainings- oder Lernphase sollte der Systemverantwortliche mit Bedacht vorgehen, was ein gewisses Know-how über Gefahren und Angriffsszenarien voraussetzt. So ist es meistens eine schlechte Entscheidung, Regeln global außer Kraft zu setzen, nur weil man den Aufwand scheut, für jedes einzelne Modul einer Anwendung zu entscheiden, ob die Regel greifen soll oder nicht. Der Aufwand fällt mit jeder Versionsänderung der Anwendung erneut an, bildet aber einen weiteren elementaren Baustein in der Absicherung. Lohn der Mühe ist dafür ein genaueres Protokoll der Anfragen, und man kann im Falle eines Falles womöglich mehr Informationen zu einem Angreifer sammeln als ohne den Einsatz der WAF.

Konfiguration – der Teufel steckt im Detail

Ein konkretes Beispiel für die Notwendigkeit einer detaillierten Konfiguration der WAF ist der Einsatz der beliebten Datenbanksoftware phpMyAdmin. Der Zugriff darauf sollte nur für den Datenbankadministrator und Entwickler freigegeben sein. Eine Freigabe von SQL-Befehlen für den gesamten Webserver kann zu Sicherheitslücken führen. Damit phpMyAdmin benutzbar ist, müssen standardmäßig unzulässige Abfragen erlaubt werden, die in der realen Anwendung nicht gestattet sein sollten, etwa *DROP*- und *ALTER-TABLE*-Befehle.

Hier bietet es sich an, virtuelle Hosts für verschiedene Bereiche (Anwendungs-Frontend, Anwendungs-Backend,

Anzeige

© Copyright by Heise Zeitschriften Verlag GmbH & Co. KG. Veröffentlichung und Vervielfältigung nur mit Genehmigung des Heise Zeitschriften Verlags.

Datenbank-Frontend et cetera) zu definieren und für jeden eine eigene WAF-Einstellung vorzunehmen. Eine Konfiguration, die sich auf den „kleinsten gemeinsamen Nenner“ bezieht, ist unsicher, und eine zu restriktive Konfiguration schränkt die Benutzbarkeit des Backends oder von Tools wie phpMyAdmin stark ein.

Relativ neu im Reigen der Sicherheitswerkzeuge für Webanwendungen sind sogenannte Database Firewalls (DBF). Sie funktionieren ähnlich wie Web Application Firewalls und werden auf der Datenbankebene zwischen Anwendung und Datenbank platziert, sodass jede Abfrage erst die Firewall passieren muss. Sie ersetzt allerdings kein Datenbank-Sicherheitskonzept mit privilegierten und weniger privilegierten Benutzerkonten, Rollenkonzepten und die Absicherung von Benutzereingaben zur Vermeidung von SQL-Injections mittels Prepared Statements und Stored Procedures.

Darüber hinaus hat eine DBF aber auch ihre Daseinsberechtigung, weil sie zusätzlichen Schutz gegen manipulierte Abfragen liefert. Konkret prüft sie anhand eines Punktesystems, ob SQL-Abfragen gefährlich sind, weil sie zum Beispiel immer wahre Aussagen („OR 1“), Kommentarzeichen, Platzhalter im Passwortfeld („password LIKE ‚%‘“) oder weitere verdächtige Teile enthalten. Jüngst entdeckten Hacker bei einem sozialen Netzwerk für Schüler, dass die Passwortüberprüfung Platzhalter zuließ. Angreifer konnten sich so Zugriff auf jedes Nutzerprofil verschaffen. Die Betreiber nahmen die Seite bis zur Schließung der eklatanten Lücke vom Netz – was offenbar noch nicht erfolgte, bis Redaktionsschluss war sie seit mehr als zwei Wochen offline. Die Database Firewall kann über die Einstufung von „normal“ und „ungewöhnlich“ in der Trainingsphase ein detailliertes Protokoll anfertigen, an welcher Stelle möglicherweise Datenlecks entstanden sind, noch bevor diese vollständig ausgenutzt werden können.

Insbesondere bei größeren Anwendungen sollte also eine Database Firewall in der Sicherheitskette (siehe Abbildung) nicht fehlen, auch wenn vermutlich der Datenbankadministrator von sich behauptet, über Berechtigungen die Datenbank ausreichend schützen zu können, und der Entwickler glaubt, seine Anwendung könne niemals unzulässige Abfragen übermitteln. Zusätzlich zu den Möglichkeiten, die Firewalls an die eigene Anwendung anzupassen, bringen sowohl WAF als auch DBF meist im Praxiseinsatz geprüfte Regeln mit, die einen gewissen Standardschutz liefern und vermutlich einen größeren Umfang abdecken, als der einzelne Administrator in der Lage ist, sich auszudenken.

Nicht totzukriegen: DoS-Angriffe

Angriffsszenarien im Betrieb einer Webanwendung, die sich leider weiterhin einer gewissen Beliebtheit bei Angreifern erfreuen, sind Denial-of-Service- und Bruteforce-Angriffe. Denial-of-Service (DoS) sollte vor allem auf Netzwerkebene verhindert werden, indem man verdächtige Angreifer aussperrt oder zumindest in der Zahl ihrer Abfragen begrenzt. Aber auch aus Entwicklersicht kann man etwas tun, um den daraus entstehenden Schaden zu begrenzen.

Sinnvoll ist etwa, den Zugriff auf rechen- oder speicherintensive Funktionen zu reglementieren. Bietet die Anwendung eine Schnittstelle nach außen an, reicht schon ein von einem unerfahrenen Programmierer geschriebenes Skript auf der Gegenseite, um massenhafte (zulässige) Anfragen bis zum Kollaps des Servers zu erzeugen. Als Gegenmaßnahme bietet sich an, API-Schlüssel zu vergeben und einen Grenzwert festzulegen (zum Beispiel maximal drei Anfragen pro Sekunde von einer IP-Adresse), den die Server- oder Web-Application-Firewall erzwingt.

Bruteforce-Angriffe zeichnen sich durch eine ähnliche Vorgehensweise aus, zielen im Gegensatz zu DoS-Angriffen aber nicht darauf, Anwendung oder Server lahmzulegen, sondern durch massenhaftes Ausprobieren Zugangsdaten zu erraten und sich so unberechtigten Zugriff zu verschaffen. Auch hier lässt sich durch eine Limitierung der Abfragehäufigkeit etwas erreichen. Zumindest, dass ein vollständiges Durchprobieren unrentabel wird, weil es zu lange dauern würde.

Andererseits benutzen Angreifer immer häufiger Botnetze, die von verschiedenen IP-Adressen simultan Angriffe im Rahmen der Begrenzung ausführen können, ohne blockiert zu werden. Hier hilft es, die Zugriffszahlen der Anwendung ständig zu beobachten und Auffälligkeiten auf den Grund zu gehen. Wenn plötzlich die dreifache Menge an Anfragen aus der ganzen Welt eintrifft und alle auf das Login-Formular abzielen, dann stimmt etwas nicht.

Nicht zu unterschätzen beim Betrieb einer Webanwendung sind darüber hinaus Angriffe von sogenannten Innentätern. Diese kennen meist die Struktur von Hard- und Software genau, entwickeln vielleicht an der Anwendung und spähen Daten aus, um sich einen Vorteil zu verschaffen. Klassische Schutzmechanismen wie Passwort- oder IP-Sperre wirken in diesen Fällen nicht, weil das Passwort für den *root*-Benutzer bekannt ist oder der Zugriff aus dem internen Netz kommt.

Der Feind von innen

Hier hilft lediglich, neben einer entsprechenden juristischen Absicherung über den Arbeitsvertrag eine „Need-to-know-Basis“ einzuführen. Ihrzufolge erhält jeder Mitarbeiter nur genau die Informationen und Zugriffe, die er auch wirklich für seine Arbeit benötigt. Administrator-Passworte dürfen dann nicht mehr „ausgeliehen“ und Berechtigungen nicht temporär gelockert werden. Selbst wenn dies womöglich den Arbeitsfluss behindert oder eine Anpassung der Prozesse erfordert, sollte die jeweilige Anfrage nur der zuständige Mitarbeiter bearbeiten, der als zusätzliche Kontrollinstanz dient. Eine weitere wichtige organisatorische Maßnahme ist das Mehraugenprinzip.

Diese Maßnahmen räumen zwar nicht alle Missbrauchsmöglichkeiten aus, denn schließlich benötigt jeder Mitarbeiter gewisse Daten für seine Arbeit, aber zumindest schränken sie den Vollzugriff ein. Gleichzeitig kann eine Protokollierung der Datenzugriffe – im Rahmen der rechtlichen und arbeitsvertraglichen Zulässigkeit – helfen, interne Datenlecks auch später noch aufzuspüren. (ur)

RENÉ KELLER

realisiert sichere Webanwendungen für die Explicatis GmbH.

JÖRN WAGNER

arbeitet an automatischen Layouts bei der medieninnovation.com GmbH.

MARTIN WUNDRAM

untersucht Webanwendungen auf ihre Sicherheit bei der Tronicguard GmbH.

© Copyright by Heise Zeitschriften Verlag GmbH & Co. KG. Veröffentlichung und Vervielfältigung nur mit Genehmigung des Heise Zeitschriften Verlags.