Bundesamt
für Sicherheit in der
Informationstechnik

Dokument ist noch aktuell. (Stand 2020)

Sicherheit von Webanwendungen

Maßnahmenkatalog und Best Practices

Im Auftrag des

Bundesamtes für Sicherheit
in der Informationstechnik

erstellt von:

SecureNet

Version 1, August 2006

Bundesamt für Sicherheit in der Informationstechnik

Godesberger Allee 185-189, 53175 Bonn ▪ Postfach 200363, 53133 Bonn
Tel.: 01888 9582-0 ▪ +49 (0)3018 9582-0
Fax: 01888 9582-5400 ▪ +49 (0)3018 9582-5400
Internet: www.bsi.bund.de

Bundesamt für Sicherheit
in der Informationstechnik
Postfach 20 03 63
53133 Bonn
www.bsi.bund.de

SecureNet GmbH
Münchner Technologiezentrum
Frankfurter Ring 193a
80807 München
www.securenet.de
Thomas Schreiber
Achim Hoffmann

Summary

Der vorliegende Maßnahmenkatalog stellt Schutzmaßnahmen und Best Practices zur Vorbeugung gegen typische Schwachstellen in Webanwendungen bereit. Ein vorangestellter Leitfaden gibt Hinweise für ein systematisches Vorgehen zur Erstellung sicherer Webanwendungen. Dabei werden sowohl bereits bestehende, als auch neu zu entwickelnde Webanwendungen betrachtet.

Der Maßnahmenkatalog richtet sich an Projektleiter und Softwareentwickler, die Webanwendungen konzipieren und implementieren.

Einleitung

Browser und Web sind heute allgegenwärtig. Nicht nur im Consumer-Bereich ist *das Web* unverzichtbares Medium für eine Vielzahl von Anwendungszwecken geworden. Auch Geschäftsprozesse zwischen Geschäftspartnern (*B2B*) sowie zwischen Bürger und Behörde (*E-Government*) werden immer stärker im Web abgewickelt. Die Spannweite reicht von einfachen Anwendungen wie der Suche nach Produktinformationen oder dem Bezug von Formularen, über Auktionen, bis hin zu Produktbestellung, Internet-Banking oder Abwicklung von Ausschreibungen, und nicht zuletzt dem Zugang ins unternehmenseigene Intranet.

Angetrieben wird diese rasante Entwicklung von den enormen Möglichkeiten, Geschäftsprozesse zu vereinfachen, zu beschleunigen und deren Produktivität zu erhöhen. Die Nutzung des Web führt zu Kosteneinsparungen, verschafft Wettbewerbsvorteile und eröffnet neue Geschäftsfelder.

Im Bestreben, diese Vorteile bestmöglich zu nutzen, werden die Augen vor den Risiken all zu häufig verschlossen: Methoden, die sich in anderen Bereichen der Informationstechnologie bewährt haben, werden ohne ihre Eignung und ihre Sicherheit zu hinterfragen ins Web übernommen; sensible Daten und Verbindungen zum Backend werden ohne zusätzliche Sicherungen über das Web zugänglich gemacht – häufig in der Annahme, der erforderliche Schutz sei durch die Netzwerk-Firewall gegeben; Webanwendungen werden in den Produktivbetrieb übernommen, ohne dass sie strenge Qualitätskontrollen in Bezug auf Sicherheitseigenschaften durchlaufen müssen; unternehmenskritische Geschäftsprozesse werden direkt auf das Web abgebildet, ohne ihre Angreifbarkeit zu untersuchen und Schutzmaßnahmen vorzunehmen.

Dabei stellt gerade das Web besonders hohe und im Vergleich zu anderen Bereichen der IT ganz spezielle Anforderungen an die Sicherheit:

- Die Offenheit des Web macht eine Webanwendung nicht nur für den legitimen Benutzer mühelos zugänglich, sondern potentiell auch für jeden nicht-legitimierten "Interessenten" erreichbar.

- Technologien wie Firewalls, die Schutz vor Angriffen an zentraler Stelle herstellen, sind auf der Ebene des Web in Form so genannter WebShields oder Application Firewalls zwar mittlerweile auch verfügbar, kommen aber in der Stärke des Schutzes den Firewalls nicht gleich und werden gegenwärtig noch wenig eingesetzt.

- Das Web bietet eine Reihe von Möglichkeiten, anonym zu agieren. Das Risiko eines Angreifers ist daher gering und die Hemmschwelle entsprechend niedrig. Hinzu kommt, dass Angriffe im Web oftmals weniger Vorkenntnisse erfordern als auf der Netzwerkebene.

- Das Web ist in vielen Fällen direkter Umsatzträger und hat dann einen großen Einfluss auf den Geschäftserfolg eines Unternehmens. So steht die Forderung einer möglichst uneingeschränkten Nutzung im ständigen Zielkonflikt mit den Erfordernissen der Sicherheit.

- Wegen seiner einfachen Zugangsmöglichkeiten wird das Web auch von Anwendern genutzt, die nicht die erforderlichen Kenntnisse besitzen, um sich angemessen zu schützen.

Im Zuge von Konzeption und Implementierung von Webanwendungen wird Sicherheitsaspekten vielfach nur unzureichende Beachtung geschenkt: Richtlinien zur sicheren Programmierung werden nicht vorgegeben oder beachtet; Freigabeprozesse, die vor der Produktivschaltung ein definiertes Sicherheitsniveau herstellen würden, existieren nicht; generell verfügbare Schutzmaßnahmen werden zugunsten maximaler Verfügbarkeit und Kostenersparnis nicht angewandt. Und schließlich: Softwareentwicklungs-Verträge und -Budgets sehen die Sicherheit der Anwendung als Teilaspekt des Gesamtprojektes oft nicht vor.

Die vorliegende Zusammenstellung von Best Practices enthält vielfach erprobte Maßnahmen zur Herstellung sicherer Webanwendungen. Sie soll technisch orientierten Lesern, insbesondere auch Softwareentwicklern als Anleitung und zum Nachschlagen dienen.

Der Themenbereich "Sicherheit von Webanwendungen" wird in einem 5-Ebenen-Modell dargestellt. Dieses enthält neben der Systemebene, der Technologieebene und der Implementierungsebene auch die Ebenen der Anwendungslogik und der Semantik. Die beiden letzteren Ebenen sind dabei in der Regel bei der verantwortlichen Fachstelle angesiedelt, so dass sich der Maßnahmenkatalog auch an Projektleiter im Bereich Webanwendungen richtet.

Auch wenn der Maßnahmenkatalog nicht als umfassende Anleitung für ein Sicherheitsaudit einer Webanwendung gedacht ist, kann er auch hierfür ein gutes Stück weit genutzt werden.

Während in Kapitel 1 ein Leitfaden zur systematischen Erstellung sicherer Webanwendungen vorgestellt wird, befindet sich in Kapitel 2 die Zusammenstellung der Einzelmaßnahmen.

Auf eine detaillierte Beschreibung von Schwachstellen wird bewusst verzichtet. Hierfür wird der Leser auf Quellen im Internet verwiesen.

Inhaltsverzeichnis

1 Leitfaden zur Sicherheit in Webanwendungen

In diesem Kapitel wird eine Anleitung zur Erstellung von sicheren Webanwendungen vorgestellt. Die hierfür angeführten allgemeinen Schritte und Hinweise müssen dabei im Rahmen einer konkreten Anwendung durch projektspezifische Erfordernisse ergänzt und detailliert werden. Hierzu können z.B. zählen: Architekturfragen, softwaretechnische Randbedingungen wie Technologie und Programmiersprache, individuelle Anforderungen einer Webanwendung oder Website, bestehender Finanzrahmen oder auch individuelle Prozesse für Entwicklung und Betrieb einer Anwendung. Wenngleich diese genannten Teilaspekte im Rahmen eines konkreten Projektes eine wichtige Rolle spielen, wird der Fokus hier - auch mit Blick auf eine notwendige Abgrenzung des Themengebietes - auf eine systematische Darstellung organisatorischer und technischer Schutzmaßnahmen gelegt.

In Hinblick auf teilweise enge Abhängigkeiten und Wechselwirkungen zwischen Sicherheitsaspekten in Webanwendungen und anderen Bereichen der IT-Sicherheit wird für eine organisatorische Zuordnung der für Webanwendungen relevanten Teilaspekte hier ein Ebenenmodell vorgestellt. Dieses Ebenenmodell wird zunächst kurz im nachfolgenden ersten Teilkapitel erläutert.

Ein genereller Unterschied im Lösungsansatz ergibt sich aus der Fragestellung, ob eine bereits bestehende oder eine neu zu entwickelnde Webanwendung betrachtet wird. Hieraus resultiert eine getrennte Beschreibung im zweiten und dritten Teilkapitel des Leitfadens.

1.1 Ebenenmodell zur Sicherheitskonzeption von Webanwendungen

Auf Grundlage eines Ebenenmodells lassen sich die Zuständigkeiten der relevanten Organisationsbereiche den einzelnen Teilaufgaben bei Sicherheitskonzeption und Realisierung von Webanwendungen zuordnen. Ausgangspunkt ist eine Unterteilung in 5 Ebenen:

Ebene		Inhalt (Beispiele)
5	**Semantik**	Schutz vor Täuschung und Betrug - Informationen ermöglichen Social Engineering-Angriffe - Gebrauch von Popups u.ä. erleichtern Phishing-Angriffe - Keine Absicherung für den Fall der Fälschung der Website
4	**Logik**	Absicherung von Prozessen und Workflows als Ganzes - Verwendung unsicherer Email in einem ansonsten gesicherten Workflow - Angreifbarkeit des Passworts durch nachlässig gestaltete "Passwort vergessen"-Funktion - Die Verwendung sicherer Passworte wird nicht erzwungen
3	**Implementierung**	Vermeiden von Programmierfehlern, die zu Schwachstellen führen - Cross-Site Scripting - SQL-Injection - Session Riding
2	**Technologie**	Richtige Wahl und sicherer Einsatz von Technologie - unverschlüsselte Übertragung sensitiver Daten - Authentisierungsverfahren, die nicht dem Schutzbedarf angemessen sind - Ungenügende Randomness von Token
1	**System**	Absicherung der auf der Systemplattform eingesetzten Software - Fehler in der Konfiguration des Webservers - "Known Vulnerabilities" in den eingesetzten Softwareprodukten - Mangelnder Zugriffsschutz in der Datenbank
0	**Netzwerk & Host**	Absicherung von Host und Netzwerk

Abbildung 1: Ebenenmodell

Ebene 0 – Netzwerk und Host

Die Ebene von Netzwerk, Server-Hardware und darauf laufendem Betriebssystem wird hier nicht direkt der Sicherheit der Webanwendung zugeordnet. Diese Ebene schließt sich vielmehr nach unten an. Die Umsetzung grundlegender Sicherheitsmaßnahmen auf dieser Ebene wird gleichwohl als zwingende Voraussetzung für sichere Webanwendungen betrachtet.

Ebene 1 – Systemebene

Hier werden all jene Programme betrachtet, die für das Funktionieren der gesamten Webanwendung benötigt werden. Dazu gehören der Webserver und der Applikationsserver, aber auch Datenbank- und Backend-Systeme. Diese Komponenten müssen bei der Sicherheitskonzeption einer Webanwendung mit einbezogen werden.

Ebene 2 – Technologie

Diese Ebene betrifft die Verwendung der für den jeweiligen Einsatzzweck und Schutzbedarf richtigen Technologie, sowie deren korrekte Nutzung. So z.B. setzt eine Webanwendung, die sensible Daten unverschlüsselt über das Internet transferiert, nicht die richtige Technologie ein. Eine Webanwendung, die Passworte zwar verschlüsselt, dafür aber einen zu kurzen Schlüssel verwendet, setzt die richtige Technologie falsch ein.

Ebene 3 – Implementierung

Die Implementierungsebene umfasst den Bereich der unbeabsichtigten Programmierfehler (Bugs), aber auch nicht vorhandene oder ungenügende Prüfung von Eingabedaten ("Data Validation"). Diese Ebene beinhaltet ferner ungenügende Testverfahren, und die Vernachlässigung der Qualitätssicherung zugunsten des Inbetriebnahmetermins oder aus Kostengründen.

Ebene 4 – Logik

Diese Ebene betrifft sowohl die Logik der Abläufe innerhalb einer Webanwendung, als auch die Interaktion mit dem Benutzer. Ist diese zu 'zweckorientiert' implementiert, kann gegebenenfalls eine Missbrauchsmöglichkeit vorliegen. Wird etwa zur Verhinderung einer mehrfach wiederholten Eingabe eines Passwortes nach dem fünften fehlerhaften Loginversuch die entsprechende Benutzerkennung gesperrt, so könnte dieser Benutzer durch Dritte gezielt ausgesperrt werden, sofern keine weiteren Maßnahmen getroffen werden. Diese missbräuchliche Vorgehensweise wird weiter erleichtert, wenn die Benutzerkennung einfach zu erraten ist.

Ebene 5 – Semantik

Die semantische Ebene umfasst inhalts- und kommunikationsbezogene Aspekte. Sie stellt den Vertrauenskontext für die Interaktion mit einem Benutzer her. Wird in diesem Bereich nicht ein hohes Maß an Sorgfalt aufgewendet, so kann eine Webanwendung von Dritten missbraucht werden, um einen Benutzer zu täuschen. Dieser Bereich kann selten auf eine einzelne Anwendung beschränkt bleiben. Vielmehr ist eine website- oder unternehmensübergreifende Betrachtung notwendig. Missbrauchsmöglichkeiten, die sich Fehler auf der semantischen Ebene zunutze machen, sind Social Engineering, Phishing, Identitätsdiebstahl u.a.

Die Bedeutung der logischen und semantischen Ebene wird vielfach nur unzurei-

chend wahrgenommen. Dennoch haben diese beiden Ebenen eine hohe Bedeutung, wenn man unter der Sicherheit einer Webanwendungen nicht allein nur den Schutz des Servers versteht, sondern eine umfassende Perspektive zugrunde legt: Der Anbieter einer Webanwendung trägt nicht nur Verantwortung für eigene Systeme, sondern auch für alle an der Nutzung der Webanwendung Beteiligten. Auf der Ebene der Logik und der Semantik kommt dieser Aspekt ganz besonders zum Tragen.

Im Rahmen einer Plan/Build/Run-Organisation kann nun folgende Zuordnung hergestellt werden:

Ebene		**Verantwortliche Org. inheit**	**Funktion**	**Fachkenntnisse**	**Grad der Toolunterstützung**
5	**Semantik**	Zentrale	Plan	Corporate Identity und Unternehmenskommunikation	.....
4	**Logik**	Fachstelle (Anforderer)		Kenntnisse der Geschäftsprozesse	■.....
3	**Implementierung**	Entwickler (Umsetzer)	Build	Softwareentwicklung	■■■■.....
2	**Technologie**	Fachstelle, Entwickler, Betrieb		Allg. IT-Security	■■.....
1	**System**	Betrieb	Run	Netzwerk- und Systemadministration	■■■■■■■
0	**Netzwerk & Host**				

Abbildung 2: Zuordnung der Ebenen

Die Spalte "Fachkenntnisse" zeigt für die einzelnen Bereiche, welche Kenntnisse jeweils erforderlich sind. In der letzten Spalte ist dargestellt, in welchem Umfang dabei auf Toolunterstützung gesetzt werden kann.

1.2 Leitfaden für bereits bestehende Webanwendungen

1.2.1 Zur Situation bei älteren Webanwendungen

Viele ältere Webanwendungen sind entweder ohne klare Sicherheitsanforderungen in den Produktivbetrieb übernommen worden oder zu einer Zeit erstellt worden, als das allgemeine Verständnis über die Anwendungssicherheit noch sehr gering gewesen ist. So ist die Sicherheit einer bestehenden Webanwendung häufig dadurch bestimmt, inwieweit die jeweiligen Entwickler Wert auf Sicherheitsaspekte gelegt haben oder das erforderliche Know-how vorlag – nicht aber, weil Vorgaben und Abnahmeprozesse ein definiertes Niveau hergestellt haben. Insbesondere in der Anfangszeit der Webnutzung wurde vielfach auf ein professionelles Software-Engineering verzichtet. Entsprechende "Altlasten" aus Perl- oder anderen Skripten sind heute noch anzutreffen.

1.2.2 Alle Webanwendungen müssen sicher sein

In aller Regel gilt: Je älter eine Webanwendung ist, desto weniger wird sie aktuellen Sicherheitsanforderungen gerecht. Zunächst ist daher Sorge zu tragen, bestehende Webanwendungen auf ein definiertes Sicherheitsniveau zu bringen. Dabei sollte nicht von der weit verbreiteten, aber falschen Annahme ausgegangen werden, dass eine unsichere Webanwendung dann zu keinem Schaden führt, wenn sie selbst keine sensiblen Daten bereitstellt oder keine sicherheitsrelevanten Funktionen ausführt. Denn: Eine einzige unsichere Webanwendung kann die Sicherheit weiterer Systeme kompromittieren. So z.B. könnten angeschlossene Application- oder Datenbankserver gefährdet sein.

1.2.3 Kosten-Nutzen-Abwägung

Sicherheitsfunktionen nachträglich in eine bestehende Anwendung zu integrieren, wird sich in den meisten Fällen als schwierig und teuer erweisen. Das gilt auch im Falle der Webanwendungen. Ein Programm, das Ein- und Ausgaben nicht über zentrale Schnittstellen abwickelt, ist nachträglich nur mühsam und fehlerträchtig mit Funktionen zur *Data Validation* auszustatten. Gleiches gilt, wenn beispielsweise Session-Merkmale nicht nur zur Authentisierung, sondern gleichzeitig zu anderen Zwecken genutzt werden, und dadurch beispielsweise die SessionID nach dem Login nicht einfach erneuert werden kann, um auf diese Weise Session Fixation zu verhindern.

Bei entdeckten Schwachstellen in bestehenden Anwendungen stellt sich daher regelmäßig die Frage, ob sich der Aufwand einer Behebung überhaupt lohnt. Bei der Beantwortung dieser Frage darf nicht außer Acht gelassen werden, dass eine unsichere Anwendung unter Umständen die Sicherheit weiterer Systeme kompromittieren kann. Hier ist eine Risikoanalyse unumgänglich, bei der zu ermitteln ist, ob der Verzicht auf die Behebung der Probleme ein tragbares Risiko darstellt oder aber eine Beseitigung der Probleme unverzichtbar ist. Erschwerend kann hinzukommen, dass bei bestehenden Anwendungen oftmals die ursprünglichen Entwickler nicht mehr verfügbar sind, und daher für eine erneute Einarbeitung und Analyse der Webanwendung zusätzliche Kosten entstehen.

1.2.4 Einsatz von WebShields

Eine sinnvolle Maßnahme zur zusätzlichen Absicherung älterer Webanwendungen stellen Web Application Firewalls (auch "Web Shields" genannt) dar. Die Einführung eines WebShields sollte jedoch nicht ohne eine Untersuchung der Sicherheitsfunktionen bestehender Anwendungen geschehen. Wird etwa festgestellt, dass bei einer Anwendung durch Eingabe eines Hochkommas SQL-Injection möglich ist, jedoch andere Eingaben trotz fehlender Data Validation in der Anwendung keine problematischen Auswirkungen haben, so kann ein WebShield dafür genutzt werden, das Hochkomma herauszufiltern – eine möglicherweise teure Fehlerbehebung in der Anwendung wäre damit nicht erforderlich. Wird jedoch ein WebShield ohne Betrachtung der zu schützenden Anwendungen als einzige Maßnahme eingesetzt, so kann das zu einer falschen Sicherheitseinschätzung führen: So z.B. ist das Herausfiltern von Sonderzeichen nicht immer ausreichend, um SQL-Injection zu verhindern.

Das durch ein WebShield erreichbare Schutzniveau sollte trotz des prinzipiell ähnlichen Funktionsprinzips nicht mit dem von Firewalls auf der Netzwerkebene gleichgesetzt werden.

1.2.5 Vorgehen für bestehende Webanwendungen

Folgende Schritte sollten durchgeführt werden, um bestehende Webanwendungen auf ein definiertes Sicherheitsniveau zu bringen. Einzelne Punkte davon sind auch im nachfolgenden Kapitel weiter ausgeführt:

- Absicherung der Web-Infrastruktur
  - Netzwerk absichern
  - Serverhärtung
  - Monitoren von Security Bulletins
  - Patchmanagement
- Bestandsaufnahme
  - Sicherheitsanalyse der Website als Ganzes
  - Betrachtung der Sicherheit jeder einzelnen Webanwendung
- Sicherheitsanalyse / Penetrationstest
  - Anwendungen auf das Vorhandensein von Schwachstellen untersuchen
- Risikoanalyse
- Festlegung von Schutzmaßnahmen
  - Maßnahmen und Best Practices anwenden
- Ggf. Einsatz eines WebShields

- Sicherheit der Webanwendung als Prozess etablieren bzw. in bestehenden Sicherheitsprozess integrieren

Insbesondere sollte Schritt 3 des folgenden Teilkapitels auch hier berücksichtigt werden.

1.3 Leitfaden für neu zu entwickelnde Webanwendungen

Bei der Neuimplementierung einer Webanwendung sollten Sicherheitsaspekte im gesamten Software-Entwicklungsprozess, also bereits von Beginn an, berücksichtigt werden.

Die folgenden Schritte geben Anhaltspunkte für ein strukturiertes Vorgehen zur Herstellung eines definierten Sicherheitsniveaus für neu zu entwickelnde Webanwendungen:

1.3.1 1 – Individuelle Bedrohungsanalyse

Während große und sicherheitskritische oder auch neuartige Anwendungen einer individuellen Bedrohungsanalyse unterzogen werden sollten, können für eine Vielzahl kleinerer und mittlerer Anwendungen in aller Regel allgemeintypische Bedrohungen angenommen werden, so dass hier der Aufwand für eine detaillierte Bedrohungsanalyse deutlich reduziert werden kann.

Generell wird bei einer Bedrohungsanalyse untersucht, welchen Bedrohungen die Anwendung und ein damit verbundener Gesamtprozess ausgesetzt ist. Mögliche Angriffspunkte werden identifiziert und fließen in die spätere Modellierung der notwendigen technischen und organisatorischen IT-Sicherheitsmaßnahmen ein.

1.3.2 2 – Festschreiben der Verantwortlichkeiten

Die Verantwortlichkeiten für einzelne Teilbereiche der Web-Sicherheit sind festzulegen. Als Ausgangspunkt hierfür kann Abbildung 2 (Zuordnung der Ebenen - vgl. Kapitel 1.1) herangezogen werden.

Dabei werden die anfordernde Fachstelle, die Entwicklung und der Betrieb als wesentliche Beteiligte gesehen und die Verantwortlichkeiten den Ebenen der Web-Sicherheit zugeordnet. Im Rahmen einer Plan/Build/Run-Organisation ergibt sich die in Abbildung 2 dargestellte Zuordnung zu den drei Funktionen.

Die Fachstelle ist am ehesten in der Lage, die Sicherheit auf der Semantischen Ebene und der Ebene der Anwendungslogik zu gewährleisten. Sie muss davon ausgehen können, dass die Entwicklungsstelle die Anwendung sicher implementiert hat, und dass der Betrieb die Anwendung sicher einsetzt und administriert.

1.3.3 3 – Umsetzung von Schutzmaßnahmen

Bei der Umsetzung von Schutzmaßnahmen sind typische und gegebenenfalls spezifische Bedrohungen zu berücksichtigen. Hierbei handelt es sich weitgehend um Bedrohungen, denen grundsätzlich jede Webanwendung mehr oder weniger stark ausgesetzt ist. Die Kenntnis grundsätzlicher Bedrohungen ist für die Auswahl und die Festlegung des Umfangs der anzuwendenden Maßnahmen unverzichtbar.

Im Folgenden werden die wichtigsten Maßnahmen und Herangehensweisen genannt, um eine Webanwendung sicher zu implementieren. Bei der Zuordnung der Maßnahmen zu den organisatorisch verantwortlichen Stellen helfen die bei jeder Maßnahme

angegebene Ebene und das oben unter 2 gezeigte Schema.

Systemebene – Ebene 1

Eine Auswahl hierfür relevanter Maßnahmen ist in **M335** bis **M400** zusammengefasst.

Als generelles Prinzip ist der möglichst weitgehende Aufbau einer "Second Line-of-Defense" anzuraten: Es ist sicherzustellen, dass nach Kompromittierung einer Anwendung die weiteren möglichen Auswirkungen auf den Rest des Systems minimal bleiben. Die aufgeführten Maßnahmen erhalten hierzu Hinweise.

Durch das Verfolgen von Security Bulletins, sowie eine zeitnahe Reaktion bei Bekanntwerden von Sicherheitsproblemen ist das System auf aktuellem Sicherheitsstand zu halten (**M335**).

Zusätzlich empfehlenswert ist die Berücksichtigung von Web Security Tools (**M420**).

Technologieebene – Ebene 2

Der Einsatz von SSL für Teilfunktionen oder die gesamte Anwendung ist zu prüfen (**M310**).

Das dem Schutzbedarf angemessene Authentisierungsverfahren ist zu ermitteln.

Implementierungsebene – Ebene 3

Data Validation: Die Validierung von Input und Output ist das A und O einer sicheren Webanwendung. Der Datenfluss nicht nur vom Benutzer zur Anwendung und umgekehrt, sondern auch zu den verschiedenen Subsystemen, und von dort in die Ausgabe, ist zu untersuchen und in ein Konzept zur Data Validation zu integrieren. Im Idealfall werden alle Input-/Output-Daten von einem zentralen Data Validation-Modul geprüft. Siehe **M100** bis **M150**

Session absichern: Zu den Voraussetzungen für eine sichere Implementierung des Session Management zählen detaillierte Kenntnisse über den Austausch von HTTP-Messages, über Session-Aufbau und Session-Ablauf, über den Austausch und die Speicherung von Cookies, sowie Kenntnisse spezifischer Bedrohungen. Grundlegende Maßnahmen sind in **M170** und **M200** angegeben. Abhängig vom Schutzbedarf ist die Session zusätzlich durch die in **M180** und **M190** genannten Maßnahmen abzusichern. Eine Absicherung gegen Session Riding ist bei den gegenwärtigen Technologien für kleinere Anwendungen teuer, bei großen oder sicherheitskritischen Anwendungen fällt sie weniger ins Gewicht. **M220** ist bei entsprechendem Schutzbedarf anzuwenden.

Weitere grundlegende Maßnahmen, die sich oft mit geringem Zusatzaufwand umsetzen lassen, sind:

- Datenbankzugriff mit *Prepared Statements* oder durch Einsatz von OR-Mappern ausführen (**M140**)
- Weiterleitungen (Redirects) niemals unkontrolliert ausführen (**M160**)
- Durch eine entsprechende Architektur des Models (Model-View-Controller) Privilege Escalation verhindern (**M225**)

Logische Ebene – Ebene 4

Die Fachstelle hat die Prozesse so zu gestalten, dass sie an keiner Stelle angreifbar sind. Dabei ist auf Konsistenz zu achten und der Prozess als Ganzes zu betrachten.

Insbesondere sind Vorkehrungen gegen Überflutung und Enumeration zu treffen. Hier muss die Fachstelle der Entwicklungsstelle Vorgaben in Form von Grenzwerten und Verfahren machen, die sich aus der Ausprägung des jeweiligen Geschäftsprozesses ergeben. (**M270**, **M280**)

Semantische Ebene – Ebene 5

Falls noch nicht geschehen, so sind auf dieser Ebene unternehmensweite Regelungen und Konventionen zu schaffen. Dies sollte unter Einbeziehung zentraler Stellen wie der Unternehmenskommunikation und des Marketing erfolgen (**M260**).

1.4 Verweise

Weitere umfassende Quellen zur Sicherheit von Webanwendungen sind frei im Internet verfügbar, so z.B.:

- OWASP Guide to Building Secure Web Applications and Web Services. Version 2.0 [1] und Version 3.0 Working Draft [2]
- OWASP Web Application Penetration Checklist [3]
- The OWASP Testing Project [4]

2 Maßnahmen und Best Practices

In diesem Kapitel werden Schutzmaßnahmen und Best Practices zur Vorbeugung gegen typische Schwachstellen in Webanwendungen vorgestellt. Die tatsächliche Umsetzung einzelner Maßnahmen steht dabei in enger Abhängigkeit von einem konkreten Projekt. So können sich bestimmte, in den meisten Fällen sinnvolle Maßnahmen im Einzelfall als unpraktikabel erweisen. Diesbezüglich ist daher eine geeignete Auswahl zu treffen.

2.1 M100 Data Validation: Filterung

Abstract **Input und Output einer Webanwendung sind zu validieren und zu filtern.**

Ebene **Implementierung**

Beschreibung

Alle Daten, die von außen in die Anwendung gelangen, sind zu validieren und zu filtern. Neben den offensichtlichen Eingabedaten in Form-Variablen existieren eine Reihe weiterer Quellen. Ebenso sind Ausgaben an den Browser zu filtern, wenn dies nicht bereits durch die Inputfilterung mit abgedeckt ist. Der Abschnitt "Output Filterung" geht auf diese Frage näher ein.

2.1.1 Input- versus Output-Validierung

Eine *Input-Validierung* hat stattzufinden, bevor die Input-Daten zum Zugriff auf Subsysteme (Datenbanken, Shell, usw.) genutzt werden. Dies geschieht im Wesentlichen zur Verhinderung von Injection-Angriffen. Eine *Output-Validierung* hat vor der Ausgabe der Daten in eine Antwortseite stattzufinden. Dies geschieht im Wesentlichen zur Verhinderung von Cross-Site Scripting (XSS). Findet die Input-Validierung in dem Bewusstsein statt, dass die Daten auch für die Ausgabe zu validieren sind, kann in den Fällen, wo die Herkunft der Daten klar ist – also beispielsweise dann, wenn die Daten alleine über die Form-Variablen in die Anwendung gelangt sind – die Output-Validierung auch mit der Input-Validierung zusammen erfolgen.

2.1.2 Filterung

Filterung ist immer dann angebracht, wenn keine Seiteneffekte zu befürchten sind oder wenn die Art der Datenverwendung gefährliche Zeichen oder Zeichenketten aufgrund ihrer Natur erlauben muss. Ein Forum, in dem über JavaScript diskutiert wird, kann nicht die Eingabe von `<script>` verbieten, muss dieses aber als HTML-Tag invalidieren.

Bei der Filterung ist mit großer Umsicht vorzugehen. Allzu leicht kann eine Variante übersehen werden.

Problematisch ist z.B. auch der auf den ersten Blick vernünftig erscheinende Filter (hier in Perl RegEx-Syntax geschrieben), um `<script>`-Tags im Eingabestrom zu löschen:

```
s/<script>//g;
```

Dieser kann jedoch mit einer geschachtelten Eingabe umgangen werden. Es ist daher rekursiv zu filtern oder gegebenenfalls der Input abzulehnen.

`Input-Filterung`

Die Input-Validierung sollte grundsätzlich nach der *vollständigen* Dekodierung der vom Browser erhaltenen Daten geschehen. Der Webserver übernimmt diese Aufgabe nicht. Zu den verschiedenen Web-Programmier- und Skriptsprachen existieren je-

weils Bibliotheken, die solche Dekodierfunktionen enthalten. In einem Servlet-Container geschieht dies bei Verwendung der `getParameter()`-Methode automatisch.

Im Folgenden wird davon ausgegangen, dass die Eingabedaten bereits in der kanonischen Form vorliegen. Die Output-Validierung wird konzeptionell von der Input-Validierung separiert, auch in den Fällen, wo sie ebenfalls beim Einlesen der Daten durchgeführt werden kann. So bleibt die Input-Validierung auf die Erkennen derjenigen Zeichen beschränkt, die beim Zugriff auf Subsysteme potentiell gefährlich werden könnten (Blacklist-Ansatz), bzw. die als erlaubte Zeichen gelten (Whitelist-Ansatz). Die Zeichentabelle ist für jedes Subsystem und die jeweilige Art der Verwendung anzupassen. Grundsätzlich gilt:

- Whitelisting ist dem Blacklisting vorzuziehen, wann immer möglich.
- Beim Whitelisting ist zunächst von einer möglichst kleinen Zeichenmenge auszugehen, die dann individuell erweitert wird.
- Einfache Filter verwenden; komplexe Filter durch sequentielle Verwendung einfacher Filter nachbilden.

Die folgende Liste enthält potentiell gefährliche Zeichen, nach Subsystem geordnet:

SQL

```
'    (String)
%    (Wildcard)
_    (Wildcard)
[    (Escape-Zeichen in MS SQL)
)    (Verknüpfung)
(    (Verknüpfung)
@    (Funktion oder Variable, min. in MS SQL)
;    (Kommandokonkatenation)
+    (Textkonkatenation)
=    (Vergleich)
<    (Vergleich)
>    (Vergleich)
#    (Kommentar)
--   (Kommentar)
/*   (Kommentar)
\0
\r
\n
\t
\h
```

Systemaufrufe

Je nach Betriebssystem sind für Systemaufrufe folgende Metazeichen als potentiell problematisch einzustufen:

```
'    (Parameterübergabe)
"    (Parameterübergabe)
`    (Kommandoaufruf)
|    (Kommandoaufruf, Pipelining)
>    (Redirection, Ausgabe)
<    (Redirection, Eingabe)
*    (Wildcard)
?    (Wildcard)
;    (Kommandokonkatenation)
$    (Variablennamen)
```

```
.    (Path Traversal)
!
&    (Kommandokonkatenation)
(
)
\0
\r
\n
```

LDAP

Werden die Daten als Anfrage für ein LDAP System benutzt, dann sind folgende Zeichen als potentiell problematisch einzustufen:

```
*
=
(
)
|
&
"
```

LDAP Metazeichen:

```
;
<=
<=
~=
:
```

Sonstige

alle nicht druckbaren Zeichen von Hex-0 bis Hex-19, insbesondere

das Null-Byte `\0`, URL-enkodiert: `%00`

Carriage Return (CR), `\r`, URL-enkodiert: `%0a`

Linefeed (LF) `\n`, URL-enkodiert: `%0d`

Tabulator `\t`, URL-enkodiert: `%09`

Output-Filterung

Die Output-Filterung in Richtung Web-Browser muss sicherstellen, dass der Zeichenstrom keine Zeichen oder Zeichenketten enthält, die der Browser als Code (HTML, JavaScript, ActiveX, usw.) interpretieren würde. Diese Anforderung ist am einfachsten dadurch zu erfüllen, dass die Zeichen, die als Code interpretiert werden könnten oder in den "Codemodus" umschalten, als solche invalidiert werden. Dieses Ziel kann durch Umwandlung in die Darstellung als sog. *Named Character Reference* erreicht werden. Auch wenn die Entscheidung, welche Zeichen umzuwandeln sind, von den individuellen Verhältnissen abhängt, ist die Ersetzung dieser 5 Zeichen in der Regel ausreichend:

```
&  ==>  &amp;

<  ==>  &lt;

>  ==>  &gt;
```

```
"  ==>  &quot;

'  ==>  &#39;
```

Kommt also beispielsweise die Zeichenkette `<script>` über die Eingabe in den Ausgabedatenstrom, so wird sie zu `&lt;script&gt;` invalidiert und im Klartext angezeigt anstatt als Beginn eines Skriptes interpretiert.

Weitere Details zur Output-Filterung, insbesondere HTML-Encoding und URL-Encoding, finden sich in **M150.9** "Ausgabedaten filtern".

Zu beachten ist insbesondere auch, dass immer dann, wenn Parameter in den Response-Header geschrieben werden, diese auf das Vorhandensein von CRLF-Zeichen zu filtern sind, um Header-Manipulation zu verhindern.

Anmerkung

Vielfach wird das Klammern von Benutzereingaben mittels des `<pre>`-Tags als Lösung empfohlen. Dieser Ansatz verhindert aber nicht, dass die geklammerten HTML-Tags interpretiert werden und sollte daher nicht zum Einsatz kommen.

Zu beachten ist weiter, dass das `&`-Zeichen zuerst ersetzt wird, da es bei allen folgenden Ersetzungen wieder als Meta-Zeichen gebraucht wird. Gleiches gilt für das `;` wenn dieses ersetzt wird.

2.2 M110 Data Validation: Whitelisting statt Blacklisting

Abstract **Wann immer möglich, ist Filterung nach dem Whitelist-Verfahren dem Blacklist-Verfahren vorzuziehen.**

Ebene **Implementierung**

Beschreibung

Eine grundsätzliche Entscheidung ist die Wahl des Validierungs- bzw. Filterprinzips: Blacklisting oder Whitelisting. Beim Blacklisting werden die Muster definiert, die aus dem Eingabestrom herausgefiltert werden, alles andere wird durchgelassen. Beim Whitelisting ist es umgekehrt: was nicht ausdrücklich erlaubt ist, ist verboten. Blacklisting ist dann problematisch, wenn als "nicht kritisch" erkannte Zeichen im Verlaufe der weiteren Verarbeitung zu einem ungewollten Systemverhalten führen.

Werden problematische Muster vergessen oder nicht erkannt oder kommen neue problematische Muster hinzu, so werden sie beim Whitelist-Ansatz hingegen automatisch geblockt. Beim Blacklist-Ansatz werden sie solange durchgelassen, bis die Regeln angepasst werden. Ein Beispiel für letzteren Fall ist der Fortschritt in den Internet-Standards. Die Einführung von HTML 4.0 hat gegenüber der Vorgängerversion einige neue Elemente gebracht, die als problematisch angesehen werden können. Eine mit Blacklisting geschützte Anwendung ist hiervon nach der Einführung von HTML 4.0 solange betroffen, bis die Blacklist angepasst worden ist.

Blacklist- und/oder Whitelist-Verfahren werden häufig nicht nur mit festen Schlüsselwörtern benutzt, sondern auch als Reguläre Ausdrücke realisiert.

Generell ist das Whitelist-Verfahren dem Blacklist-Verfahren vorzuziehen.

Beispiel

Der folgende Test in Perl soll für einen Parameterwert alle Buchstaben oder Zahlen erlauben:

```
$wert = s/[^a-zA-Z0-9]//g;
```

Anmerkung

1: das Negationszeichen `^` in der Zeichenklasse erweckt zunächst den Eindruck, dass es sich hier um eine Blacklist handelt. Tatsächlich ist der Befehl so zu lesen: "lösche alle Zeichen außer Buchstaben und Zahlen".

2: in dem Beispiel werden explizit alle Zeichen (z.B. `a` bis `Z`) aufgezählt, und keine vordefinierten Zeichenklassen (z.B. `\w` oder `\d`) benutzt. Das ist wichtig, da solche Zeichenklassen nicht der Kontrolle des Programmierers unterliegen. Sie könnten sich zum Beispiel ändern, wenn eine Anwendung von einer Plattform auf eine andere portiert wird oder sich die Version der Programmiersprache (hier Perl) ändert.

Ein Blacklist-Filter wäre z.B.:

```
$wert = s/(script|object)/$1_tag/ig;
```

Diese Blacklist wird die HTML-Tags `OBJECT` und `SCRIPT` ersetzen. Unberührt davon

bleibt z.B. das `EMBED`-Tag, welches ähnlich problematisch sein kann wie das `SCRIPT`-Tag.

Bemerkung

Bei der Implementierung gilt ebenso, wie schon in **M100** erwähnt, dass man besser mehrere einfache Filter nacheinander verwendet, statt einen einzigen, sehr komplexen Filter zu konstruieren.

2.3 M120 Data Validation: Behandlung von manipuliertem Input

Abstract **Manipulierter Input ist generell abzulehnen. Weitere Hinweise über die Ursache des Fehlers sind nicht zu geben.**

Ebene **Implementierung, Logik**

Beschreibung

Bei der Behandlung von ungültigem Input kann zwischen zwei Fällen unterschieden werden: 1) Fehleingaben des Benutzers, und 2) bewusste Manipulation. Im ersten Fall ist in benutzerfreundlicher Weise zu reagieren – unter Wahrung des Minimalitätsprinzips, vgl. **M250**, **M320.** Der Benutzer ist auf den Fehler hinzuweisen und bei der Korrektur entsprechend zu führen.

Im zweiten Fall sollte die Reaktion in geeigneter Weise den Manipulationsversuch abwehren. Dann ist zu entscheiden, ob eine Filterung erfolgt oder ob die Weiterverarbeitung abgelehnt wird. Eine erfolgreiche Filterung könnte eine Weiterverarbeitung implizieren.

Ein sicheres Kriterium für den Abbruch der Verarbeitung mit einer Fehlermeldung ist immer dann gegeben, wenn die Eingabedaten mit bestimmungsgemäßer Browserbedienung nicht eintreten können, wie z.B.:

- Zusätzliche oder fehlende Form-Variablen
- Das Session-Cookie enthält Zeichen, die von der Anwendung nicht vergeben werden oder es entspricht nicht der definierten Länge.
- In `HIDDEN`-, `SELECT`- oder `CHECKBOX`-Variablen transportierte Werte wurden verändert oder entsprechen nicht dem Muster, welches die Anwendung gesetzt hat.
- Die Quelle der Variablen (z.B. GET, POST, Cookie) stimmt nicht mit der Vorgabe der Anwendung überein.

Zu den möglichen Reaktionen auf derartige Fehler können je nach Situation zählen:

- Die weitere Verarbeitung ist zu stoppen. Es ist zu erwägen, statt einer Fehlermeldung eine Reaktion in dieser Weise vorzunehmen: Dem Benutzer – dem in diesem Fall ein Angriff unterstellt wird – wird der korrekte Umgang mit der Eingabe vorgetäuscht, um seinen Angriffsversuch zu unterlaufen und weiteres Suchen nach Schwachstellen zu erschweren. So würde die Antwort auf das manipulierte Absenden eines Kontaktformulars beispielsweise genauso wie im korrekten Fall lauten: "Vielen Dank für Ihre Anfrage, die wir umgehend bearbeiten." (siehe **M250** "Minimalitätsprinzip").
- Es ist zur Homepage oder Sitemap oder der (neutralen) Seite, die auch bei Zugriffen auf nicht vorhandene Seiten angezeigt wird, zurückzukehren.
- Es ist eine explizite Warnung auszugeben, dass ein Angriffsversuch festgestellt worden ist und dass alle Zugriffsversuche protokolliert werden bzw. anderweitige Maßnahmen ergriffen werden.

- Zusätzlich bei bestehender Session: Der Benutzer ist auszuloggen und die Session zu invalidieren (siehe **M170.6** "Session beenden", **M170.7** "Sessions behandeln in PHP").

Nicht korrekt sind diese häufig anzutreffenden Reaktionen:

- ungefilterte Ausgabe der Fehlermeldung (liefert u. U. für einen Angriff verwertbare Informationen)

- Server Error 500 (daraus lässt sich schließen, dass die Anwendung nicht alle Fälle korrekt abfängt)

- "Die Anwendung ist momentan wegen Wartungsarbeiten nicht verfügbar" (gibt dem Angreifer u. U. einen Anreiz, die Anwendung weiter zu untersuchen, da er vermutet, dass er die Anwendung tatsächlich kurzzeitig lahm gelegt hat)

Gänzlich abzuraten ist in solchen Fällen von Versuchen, fehlerhafte Eingaben zu korrigieren. Derartige Versuche weisen das unnötige Risiko auf, doch nicht fehlerfrei zu sein und einen Angriff dadurch überhaupt erst zu ermöglichen.

2.4 M130 Data Validation: Selektive Filterung von HTML-Tags

Abstract **Sind HTML-Tags in besonderen Fällen als Eingabe erlaubt, so ist möglichst ein eigenes Markup zu verwenden.**

Ebene **Implementierung**

Beschreibung

Ausgesprochen schwierig ist die Output-Filterung, wenn begrenztes Markup seitens des Benutzers erlaubt sein soll. Dies kommt beispielsweise vor bei webbasierten Seitengeneratoren, wie einige Webspace-Provider sie anbieten oder bei Versteigerungs- und Kleinanzeigen-Plattformen, wo die Angebote seitens des Benutzers mit Bildern oder Hervorhebungen ausgestattet werden können. Dann muss der Filter in der Lage sein, erlaubte HTML-Tags von problematischen Tags zu unterscheiden. Dieser Ansatz ist jedoch mit dem hohen Risiko behaftet, doch etwas zu übersehen. Vorzuziehen ist daher der Ansatz, für das Markup des Benutzers eine eigene Markup-Sprache zu definieren. Diese wird dann von der Anwendung in die zugehörigen HTML-Tags übersetzt. Herkömmliche Tags bzw. problematische Zeichen werden nach wie vor ausgefiltert.

Ein mögliches Verfahren, wenn nur einfaches Markup zugelassen werden soll, ist die Verwendung von `{` und `}` statt < und >. Fett würde dann als `{F}Dies ist Fett{/F}` geschrieben und ein Bild würde auf diese Weise platziert:

```
{img src=/images/img.gif width=1 height=1 img}
```

Selbstverständlich darf die Umwandlung in HTML dann nicht aus geschweiften Klammern einfach spitze Klammern machen, sondern muss jedes Tag als Ganzes ansehen: `{img` nach `<img`, `img}` nach `>`, `src=`*Datei* nach `src="`*Datei*`"` (wobei *Datei* zu filtern ist), usw.

2.5 M140 Data Validation: SQL-Injection verhindern

Abstract **Statt Embedded bzw. Dynamischem SQL sind Prepared Statements oder andere sichere Techniken für den Datenbankzugriff zu verwenden.**

Ebene **Implementierung**

Beschreibung

2.5.1 Prepared Statements

Durch Verwendung von *Prepared Statements* anstatt dynamisch zusammengebauter SQL-Statements kann dem Problem der SQL-Injection aus dem Weg gegangen werden. Software-Entwickler müssen sich im Rahmen des Designs einer Funktion oder Methode exakte Klarheit darüber verschaffen, welche Struktur seine Datenbankabfrage hat bzw. was die Menge aller möglichen Abfragestrukturen ist, die in Abhängigkeit vom Input zur Ausführung gelangen können. Diese Abfragen sind in Form von Prepared Statements (also in der Regel entsprechenden If-Abfragen und dem Zusammensetzen von Abfragen) allen Bedingungen, die durch unterschiedlichen Input zu prüfen sind, Rechnung getragen werden. Es besteht also in aller Regel keinerlei Notwendigkeit, in die Formulierung des Statements zusätzlich eine Dynamik hinein zubringen, die nur mit dem Mittel der Dynamic Statements erreichbar wäre.

2.5.2 Stored Procedures

Stored Procedures haben im Hinblick auf ihre Sicherheit gegenüber SQL-Injection ähnliche Eigenschaften wie Prepared Statements. Auch hier ist zu beachten, dass bei Übergabe von unvalidierten Parametern an eine Stored Procedure möglicher problematischer Input nicht automatisch abgefangen wird. In [5] wird beschrieben, wie sich entsprechende Probleme beheben lassen.

2.5.3 SQL-Injection und Java

Der Zugriff auf SQL-Datenbanken erfolgt bei Java durch die JDBC-Schnittstelle (Java Database Connectivity). Von den jeweiligen Datenbankherstellern werden Implementationen dieses Interfaces, sog. JDBC-Treiber, zur Verfügung gestellt.

Parameter, die vom Benutzer übergeben werden, müssen validiert werden. Das kann auf folgende Weisen geschehen:

Eigene Filter-Funktion

In jeder Programmiersprache ist es möglich, durch Schreiben einer Escape-Funktion die SQL-Query-Routine abzusichern. Eine solche Routine könnte folgendermaßen aussehen:

```
public static String escapeSQL(String stringToBeEscaped)
{
    StringBuffer escapedBuffer = new StringBuffer();
```

```
    for (int i = 0; i < stringToBeEscaped.length(); i++)
    {
      char c = stringToBeEscaped.charAt(i);
      switch (c)
      {
        case '\'' :
          escapedBuffer.append("''");
          break;
        case '\0' :
          escapedBuffer.append("\\0");
          break;

        // ... hier weiteres Datenbank-spezifisches Quoting

        default :
          escapedBuffer.append(c);
      }
    }
    return escapedBuffer.toString();
}
```

Ein Nachteil dieser Maßnahme ist, dass alle problematischen Zeichen in Bezug auf die gerade benutzte Datenbank bekannt sein müssen.

Benutzung von java.sql.PreparedStatement

Die vorteilhaftere Möglichkeit im Falle von JDBC ist die Benutzung eines `java.sql.PreparedStatement`. Hierbei werden zunächst Platzhalter in einem SQL-Query[1] geschrieben und anschließend diese Parameter mit `setString()` gesetzt. Der datenbankspezifische JDBC-Treiber übernimmt dabei das richtige Kodieren des Parameters. Dies ist immer für die jeweilige Datenbank passend und zuverlässig gelöst (unterschiedliche Datenbanken haben unterschiedliches Quoting!).

Beispiel:

```
PreparedStatement preparedStatement =
jdbcConnection.prepareStatement ("select * from user where id=?");

preparedStatement.setString(1, param);

ResultSet resultSet = preparedStatement.executeQuery();
```

Benutzung von Objekt-relationalen Mapping-Tools

Heutzutage ist die Benutzung eines Objekt-relationalen Mapping-Tools (OR-Mapper) in der Java-Entwicklung sehr verbreitet. Diese Tools erlauben einen direkten Zugriff auf eine relationale Datenbank. Es ist in der Regel davon auszugehen, dass diese Tools `java.sql.PreparedStatement` benutzen. Dass dies tatsächlich so ist, muss bei der Benutzung verifiziert werden.

Oftmals lassen diese Tools bei komplexeren Abfragen eine Hintertür offen, indem ein Teil der Abfrage als SQL übergeben werden kann. In diesem Fall müssen die Parameter, die übergeben werden, zuvor richtig kodiert werden.

Vermeidung von "execute()"

Die JDBC API lässt bei den Klassen `Statement` und `PreparedStatement` die folgen-

[1] Diese SQL-Query-Routine darf keine Befehle enthalten, die dazu führen, dass sie doch wieder interpretiert wird, z.B. bei Verwendung von execSQL()

den Methodenaufrufe zu:

```
executeQuery("sql...")

execute("sql...")

executeUpdate("sql...")
```

Bei Benutzung der Klasse `Statement` sollte der Aufruf `execute()` vermieden werden, da es bei einigen Datenbanken möglich ist, ein SQL-Kommando zu beenden und ein zweites SQL-Kommando anzuhängen. Somit kann ein an sich harmloses Select-Kommando beendet werden und ein Update-Kommando angehängt werden.

Daher sollte bei einem Select-Kommando immer die Methode `executeQuery()` und bei einem Update-Kommando die Methode `executeUpdate()` benutzt werden.

2.5.4 SQL und MS SQL

MS SQL bietet selbst eine Möglichkeit, jeden Parameter als Stringliteral für ein SQL-Query zu betrachten: `Parameter collection`. Dazu sind SQL-Queries z.B. wie folgt zu konstruieren:

```
SqlDataAdapter myCommand = new SqlDataAdapter(
        "SELECT name FROM users WHERE name=@uid", myConnection);
SqlParameter parm = myCommand.SelectCommand.Parameters.Add( "@uid",
        SqlDbType.VarChar, 11);
parm.Value = Request.QueryString("par");
```

2.6 M150 Data Validation: Diverse Maßnahmen

Abstract **Weitere Maßnahmen und Tipps zur Data Validation.**

Ebene **Implementierung**

Beschreibung

Zusammenfassung wichtiger Funktionen

M150.1 Eingabeabfrage in ASP

Das ASP.Net Framework stellt viele Funktionen bereit, um die Parameter, die der Browser gesendet hat, an die Anwendung weiterzugeben. Es können Funktionen verwendet werden, die gezielt nach einem Parameter an einer bestimmten Stelle suchen (z.B. `Request.Form()`) oder eine Funktion (z.B. `Request()`), die den Parameter an üblichen Stellen sucht (siehe auch **M230** "Post erzwingen")

Grundsätzlich ist immer die Request-Funktion zu verwenden, welche nur den Parameter liefert, den man auch erwartet. Also z.B. `Request.QueryString("par")` für einen `GET`-Request oder `Request.Form("par")` für einen `POST`-Request oder `Request.Cookies("par")` oder `Request.ClientCertificates("par")` oder `Request.ServerVariables("par")` statt `Request("par")`.

M150.2 Serverseitig validieren

Alle Eingabedaten sind von der Anwendung auf dem Server zu validieren. Prüfungen auf dem Client (Browser) z.B. mittels JavaScript können allenfalls aus Gründen der Benutzerfreundlichkeit erfolgen, aber niemals um Annahmen über die Beschaffenheit der Eingabedaten sicherzustellen. Prüfungen im Browser könnten vom Benutzer abgeschaltet und beliebig manipuliert werden.

M150.3 Namen von Form-Variablen filtern

Genauso wie der Wert einer Form-Variablen kann auch ihr Name beliebig manipuliert werden. Daher sind auch die Namen von Form-Variablen der Filterung zu unterziehen.

Da alle Form-Variablen in der Anwendung bekannt sind, ist hier ein einfaches String-Mapping sehr effektiv.

Beispiel in Perl:

```
use CGI;
my $q = new CGI;
my $name = '';
$name = 'meinName' if (defined $q->param("name"));
```

M150.4 Länge/Größe der Eingabedaten begrenzen

Alle Eingabedaten sind auf ihre Größe zu überprüfen, bevor sie verwendet (insbeson-

dere kopiert) werden. Die genaue Realisierung dieser Prüfung ist von der jeweiligen Programmiersprache abhängig. Beispiele[2]:

Beispiel in Perl:

```
exit(22) if (length($parameter) > 42);
```

Beispiel in PHP:

```
if (strlen($parameter) > 42) { return; };
```

Achtung: die PHP-Funktionen geben im String enthaltene Null-Bytes direkt als solche weiter. Null-Bytes müssen **zuvor** entfernt werden. In Perl sind keine String-Funktionen bekannt, die dieses Problem haben.

M150.5 Datentyp der Eingabedaten

Welchen Datentyp die Eingabedaten haben, bestimmt zunächst die Konfiguration des Webservers. Allgemein muss die Applikation annehmen, dass die Daten binär sind. D.h. dass URL-kodierte Zeichen wie der Null-Character (`%00`) oder das Carriage Return-Linefeed (`%0d%0a`) bereits in die jeweiligen binären Zeichen umgewandelt worden sind. Die Applikation muss den Wertebereich, und damit den Datentyp, selbst überprüfen. Je nach Programmiersprache steht dazu eine Fülle von Funktionen zur Verfügung. Bei der Auswahl der Funktionen ist darauf zu achten, dass die Daten nicht verändert werden (manche Funktionen filtern derart, dass nur dem Wertebereich entsprechende Daten zurückgeliefert werden). Falsche Daten sind abzulehnen, nicht zu korrigieren.

M150.6 Filterung von Datentyp und Datengröße mittels RegEx

Mit Regulären Ausdrücken (RegEx) kann man Datentyp und Datengröße zusammen prüfen. Hier z.B. mit Perl:

```
exit(22) if ($parameter !~ /^[a-zA-Z]{1,42}$);

exit(22) if ($parameter !~ /^[a-zA-Z0-9]{1,42}$);

exit(22) if ($parameter !~ /^[a-zA-Z0-9]{2,9}@[a-zA-Z0-9\.-]{5,99$);
```

M150.7 Filterung in ASP.Net

Ab der Version 1.1 bietet das ASP.Net-Framework eigene Funktionen für die Datenprüfung an. Interessant sind hierbei `RangeValidator` und `RegularExpressionValidator`.

ASP RangeValidator:

```
<asp:RangeValidator
ControlToValidate="textbox"
MinimumValue="1"
MaximumValue="100"
Type="Integer"
```

[2] Der Einfachheit halber ist in den genannten Beispielen ein exit aufgerufen. Dies ist bei der Realisierung selbstverständlich durch eine geeignete Fehlerbehandlung zu ersetzen.

```
EnableClientScript="false"
Text="Gültige Werte von 1 bis 100!"
runat="server" />
```

Für `Type` stehen zur Verfügung:

```
Currency
Date
Double
Integer
String
```

ASP RegularExpressionValidator:

```
<asp:RegularExpressionValidator
ControlToValidate="textbox"
ValidationExpression="\d{5}"
EnableClientScript="false"
ErrorMessage="The zip code must be 5 numeric digits!"
runat="server" />;
```

Bemerkung: Die Controls setzen und liefern nur eine Fehlermeldung. Sie unterbrechen nicht den Programmfluss - das muss der Programmierer weiterhin selbst realisieren, indem die Rückgabewerte überprüft werden.

M150.8 Datenlänge begrenzen in Perl

Mit dem Perl-Modul CGI.pm kann man die Größe der POST-Daten zusätzlich begrenzen. Man beachte aber, dass der Webserver bereits alle Daten vom Browser entgegengenommen hat:

```
use CGI;
$CGI::POST_MAX          = 1024 * 1; # entsprechend anpassen
$CGI::DISABLE_UPLOADS   = 1;
```

M150.9 Ausgabedaten filtern

Für das HTML-Encoding und URL-Encoding bieten die einzelnen Programmiersprachen unterschiedliche Hilfsmittel und Funktionen an. Generell gilt, dass alle "nichtdruckbaren" Zeichen sowie die meisten Sonderzeichen durch HTML-Entities ersetzt werden müssen.

`HTML-Encoding`

Beispiele in ASP:

```
Response.Write("Wert: " + Server.HTMLEncode(Request.Form("par")));
```

Beispiel in Java:

In Java kann man die Prüf-Funktion `validHTML` und die Konvertier-Funktion `saveHTML` verwenden.

```
public class Validator // requires JDK 1.4 or higher
{
  private static final String
          DEFAULT_CHARS_TO_BE_ENCODED = "<&>\"'";
```

```
  public static String encodeHTML(String stringToBeEncoded)
  {
    return
    encodeHTML(DEFAULT_CHARS_TO_BE_ENCODED, stringToBeEncoded);
  }
  public static String encodeHTML(String charsToBeEncoded,
                                        String strToBeEncoded)
  {
    if (strToBeEncoded == null || charsToBeEncoded == null){
      return (strToBeEncoded);
    }
    Pattern pattern = Pattern.compile("[" + charsToBeEncoded + "]");
    Matcher matcher = pattern.matcher(strToBeEncoded);
    StringBuffer encodedBuffer = new StringBuffer();
    while (matcher.find()) {
      String match = matcher.group();
      matcher.appendReplacement(encodedBuffer, "&#" +
                                    match.getBytes()[0] + ";");
    }
    matcher.appendTail(encodedBuffer);
    return encodedBuffer.toString();
  }

  public static void main(String[] args)
  {
    String par = "ham & <b>eggs</b>";
    System.out.print("encodedHTML: " + Validator.encodeHTML(par));
    // --> encodedHTML: ham &#38; &#60;b&#62;eggs&#60;/b&#62;
  }
}
```

Anmerkung: Beide Funktionen arbeiten mit selbstdefinierten Expressions, für deren Korrektheit der Programmierer verantwortlich ist. Falls solche Funktionen in einer Bibliothek zur Verfügung gestellt werden, ist von beiden Funktionen abzuraten.

Beispiel in Perl:

In Perl gibt es viele Möglichkeiten, die Daten für die Ausgabe entsprechend zu kodieren. Die wichtigsten davon sind in den nachfolgenden Beispielen aufgelistet. Es wurde explizit nicht die Zeichenklasse `\W` (Perl RegEx) verwendet, um sicherzustellen, dass alle Zeichen konvertiert werden.

```
$encode_enc1 =~ s/([^a-zA-Z0-9_.-])/'%'.unpack('H*',$1)/ge;
$encode_hex1 =~ s/([^a-zA-Z0-9_.-])/'&#x00'.unpack('H*',$1).';'/ge;
$encode_hex2 =~ s/([^a-zA-Z0-9_.-])/sprintf('&#x%04x;',ord($1))/ge;
$encode_int1 =~ s/([^a-zA-Z0-9_.-])/'&#'.unpack('C*',$1).';'/ge;
$encode_int2 =~ s/([^a-zA-Z0-9_.-])/'&#'.ord($1).';'/ge;
```

Man kann auch folgende Funktion für sicheres HTML-Encoding verwenden:

```
sub saveHTML ($) {
    my $str = $_;
    $str =~ s/([^a-zA-Z0-9_.-])/sprintf('&#x%04x;',ord($1))/ge;
    return($str);
}
print "Parameter: ",saveHTML($par);
```

Das Perl-Modul `CGI.pm` bietet ebenfalls eine eigene Funktion für sicheres HTML-Encoding.

```
my $q = new CGI;
   $q->autoEscape(1);  # default
print "Parameter: ", $q->escapeHTML($par);
```

Anmerkung: Die obigen Perl-Beispiele erwecken zunächst den Eindruck, dass ein Blacklist-Verfahren verwendet wird - wovon in Maßnahme **M110** explizit abgeraten wird. Dem ist aber nicht so, denn die Prüfung lautet in Worten z.B.: "alle Zeichen **außer** a-z oder A-Z oder 0-9 oder _ oder . oder – ". Es wurde lediglich die logische Umkehrung verwendet.

URLs in HTML

URLs (Links) in der HTML-Seite müssen besonders kodiert sein. Diese Teile der Ausgabe sind also abweichend von der oben (HTML im Text) beschriebenen Methode umwandeln.

Beispiel in Java:

```
java.net.URLEncoder.encode(aURL,"ISO-8859-1");
```

Beispiel in ASP:

```
var BaseURL = "http://www.mysite.com/search2.asp?searchagain=";
Response.Write("<a href=\"" + BaseUrl
               + Server.URLEncode(Request.QueryString("par"))
               + "\">click-me</a>" );
```

Beispiel in Perl:

Perl selbst bietet dafür keine Funktion, sie kann aber einfach wie folgt definiert werden

```
sub saveURL ($) {
    my $str = $_;
    $str =~ s/([^a-zA-Z0-9_.-])/sprintf('%%%02x;',ord($1))/ge;
    return($str);
}
```

Anmerkung: die obigen Beispiele benutzen teilweise die Input-Parameter direkt, ohne Filterung. Das widerspricht der Maßnahme **M100**, in welcher auf genau diese Probleme hingewiesen wird. In den Beispielen wurde der Übersichtlichkeit halber bewusst auf eine vorherige Input-Filterung verzichtet.

M150.10 LDAP-Injection vermeiden (Java)

Parameter in Suchfiltern, die die Gültigkeit und die Absicht des Filters verletzen, müssen "escaped" werden. Von einer eigenen Escape-Funktion wird hier abgeraten, da das JNDI dafür eine spezielle Such-Methode bereitstellt, die automatisch alle speziellen Zeichen escaped. Hierbei werden der Suchmethode ein Filter-Template mit Parametern ({0}, {1}, {2}, usw.) und die Parameter als Object-Array übergeben.

Hier ist ein Beispiel, das diese Methode demonstriert:

```
String filterTemplate = "(userPassword={0})";
SearchControls searchControls = null; // default SearchControls
NamingEnumeration answer = jndiDirContext.search
          (filterTemplate, new Object[]{pw}, searchControls);
```

M150.11 Escape-Funktionen

Als Anhaltspunkt seien hier Escape-Funktionen verschiedener Programmiersprachen aufgeführt:

ASP

```
Request.Form()
Request.QueryString()
Request.Cookies()
RangeValidator()
RegularExpressionValidator()
Server.HTMLEncode()
Server.URLEncode()
```

Java

```
class java.sql.PreparedStatement
java.net.URLEncoder.encode()
```

Perl

```
CGI::autoEscape()
CGI::escapeHTML()
```

PHP

```
addslashes()
quote_meta()
mysql_real_escape_string()
strip_tags()
htmlentities()
utf8_decode()
```

2.7 M160 URL-Weiterleitungen (Redirects) kontrollieren

Abstract **Weiterleitungen sind durch geeignete Maßnahmen vor Missbrauch zu schützen**

Ebene **Semantik, Logik, Implementierung**

Beschreibung

Wir geben hierfür Maßnahmen an, die je nach Umständen und Anforderungen anzuwenden sind:

M160.1 Nur bekannte Links weiterleiten

Sind die Links, auf die weitergeleitet wird, auf bestimmte beschränkt, so ist die Weiterleitung auch nur für diese zu erlauben. Alle anderen Parameter sind abzulehnen. Das ist am besten dadurch zu erreichen, dass nicht die Ziel-Seite selbst als Parameter übergeben wird, sondern ein Index in eine Tabelle, die serverseitig gehalten wird, und in der alle in Frage kommenden URLs enthalten sind. Ein Redirect-Aufruf würde daher statt:

```
http://www.shop_abc.tld/redir?url=http://www.hersteller_xyz.tld/
```

so lauten:

```
http://www.shop_abc.tld/redir?url=5
```

wobei das Ziel `http://www.hersteller_xyz.tld/` in der Tabelle unter der Nummer 5 abgelegt ist.

M160.2 Manuelle Weiterleitung über Zwischenseite

Statt die Umlenkung direkt und für den Benutzer transparent durchzuführen, ist sie indirekt über eine Seite zu führen, die dem Benutzer angezeigt wird. Damit kann dieser den Link vor dem aktiven Klick prüfen und selbst entscheiden, ob er diesem vertraut. Ein Aufruf z.B. von:

```
https://b2b.abc-ag.tld/q=https://www.angreifer.tld/b2b-abc-ag.html
```

würde, statt direkt zum Ziel weiterzuleiten, eine Seite wie diese an den Browser zurückliefern:

```
Sie werden auf folgende externe Webseite weitergeleitet.

  https://www.angreifer.tld/b2b-abc-ag.html

Aus Sicherheitsgründen führen wir die Weiterleitung nicht automa-
tisch durch. Bitte kontrollieren Sie den Link und setzen Sie die
Weiterleitung gegebenenfalls durch Klick fort.
```

Der in diese Seite eingetragene Link hat eine *Data Validation* zu durchlaufen, damit er nicht zu einer Cross-Site-Scripting-Schwachstelle führt.

M160.3 Referrer-Test durchführen

Ähnlich wie in **M220**, kann auch hier der Test des Referrers einen gewissen Schutz bieten. Die Umleitung ist nur dann auszuführen, wenn der Referrer die URL der Seite enthält, auf der sich der Weiterleitungslink befindet. Eine XSS-Schwachstelle kann aber auch hier diesen Schutz wieder zunichte machen.

M160.4 Lokale Weiterleitung einschränken

Erfolgt die Weiterleitung auf eine Seite auf der eigenen Website, so ist vor der Ausführung zu prüfen, dass als Parameter keine URL übergeben worden ist, die auf eine externe Seite führt. Am einfachsten werden lokale Weiterleitungen ausschließlich mit relativen Pfaden durchgeführt und der Host-Teil statisch hinzugefügt.

Statt also die Weiterleitung so zu programmieren, dass ein vollqualifizierter Link

```
https://b2b.abc-ag.tld/q=http://b2b.abc-ag.tld/neu/seite5.html
```

übergeben werden muss, sollte der Parameter von dieser Form sein:

```
https://b2b.abc-ag.tld/q=/neu/seite5.html
```

und der Teil `http://b2b.abc-ag.tld` statisch hinzugefügt werden.

M160.5 Header-Manipulation verhindern

Weiterleitungen sind eine typische Quelle für Header-Manipulation-Schwachstellen. Es ist daher besonders darauf zu achten, dass dies nicht auftritt.

2.8 M170 Session Management

Abstract **Das Session Management ist eine der problematischsten Stellen für die Sicherheit von Webanwendungen. Für den Schutz der SessionID sollten daher entsprechend wirksame Maßnahmen zur Anwendung kommen.**

Ebene **Logik, Implementierung**

Beschreibung

Der Schutz der Session vor einem Eindringen lässt sich durch Berücksichtigung der im Folgenden genannten Maßnahmen deutlich verbessern. Die Beachtung dieser Maßnahmen ist daher der durchzuführende Mindestschutz.

Wenn im Folgenden von Cookies die Rede ist, dann ist damit immer ihre Eigenschaft als Träger der SessionID gemeint.

M170.1 SessionID nach Authentisierung erneuern

Nach erfolgreicher Authentisierung (Login) ist eine schon bestehende SessionID durch eine neue zu ersetzen.

In einer Java Servlet-Umgebung ist eine Änderung der SessionID in direkter Weise nicht möglich. Stattdessen kann folgende Variante gewählt werden: Sessiondaten speichern – alte Session invalidieren – neue Session erstellen – gespeicherte Daten in neue Session kopieren. Im Detail:

1. Nach erfolgreichem Login sind die Daten aus der bestehenden Session zu speichern

```
String savedUrl = (String)
session.getAttribute("SESSION_REDIRECT_URL");
```

2. Dann ist die bestehende Session zu invalidieren

```
session.invalidate();
```

3. Schließlich ist eine neue Session zu erstellen – damit wird gleichzeitig eine neue SessionID vergeben

```
session = request.getSession();
```

4. Und die gespeicherten Daten in die neue Session zu kopieren.

```
if ( savedUrl != null)
   session.setAttribute("SESSION_REDIRECT_URL", savedUrl);
```

5. Jetzt ist der Loginvorgang beendet und der Benutzer setzt die Bearbeitung mit einer neuen SessionID fort.

M170.2 Cookies einschränken

Cookies sind, wenn sie Träger von sensitiven Informationen sind (insbesondere der

SessionID), auf einen minimalen "Aktionsradius" zu beschränken, um die Gefahr zu minimieren, dass sie von nicht vertrauenswürdiger Seite aufgefangen werden.

Eine *gute* Cookie-Definition sieht so aus:

```
Set-Cookie: SESSIONID=iojew984389hui78g; path=/appl/portal; secure;
HttpOnly
```

Der Pfad ist auf die Anwendung einzuschränken, `secure`- und `HttpOnly`-Flag (siehe unten) sind zu setzen.

Damit diese Pfad-Einschränkung möglich ist, sind alle an einer Anwendung beteiligten Ressourcen unterhalb des Einstiegspfades zu platzieren.

Domain-Cookies sollten grundsätzlich vermieden werden. (z.B. `domain=.meinedomain.tld`, d.h. Domain-Cookies werden an jeden Host in der Domain geschickt). Architekturen, die Domain-Cookies benötigen, sind sehr genau auf die Auswirkungen auf die Sicherheit zu untersuchen.

M170.3 secure-Flag setzen

Wann immer eine Anwendung SSL-Verschlüsselung benötigt, ist das Cookie mit dem `secure`-Flag zu definieren. Damit wird sichergestellt, dass der Browser das Cookie niemals über eine unverschlüsselte Verbindung überträgt. Beispiel:

```
Set-Cookie: SESSIONID=lk8usdjiui090iwef; path=/myApp; secure
```

M170.4 HttpOnly-Flag setzen

Im Cookie ist das `HttpOnly`-Flag zu setzen. Es verhindert beim Internet Explorer ab Version 6 SP1, dass das Cookie von JavaScript etc. gelesen werden kann. Eine XSS-Lücke kann daher nicht mehr zum Auslesen des Cookies genutzt werden.

Die Verwendung ist jedoch nicht unproblematisch: Die wichtigsten aktuellen Browser neben dem Internet Explorer akzeptieren zwar Cookies mit dem `HttpOnly`-Flag, unterstützen aber dessen Funktionalität nicht, d.h. hier verhält es sich wie ein ganz normales Cookie, kann also nach wie vor von JavaScript etc. gelesen werden. Bei manchen älteren Browsern führt die Angabe dieses Flags zudem zu einem Syntaxfehler und damit zur Ablehnung des Cookies. Damit das Session Management auch bei Verwendung des `HttpOnly`-Flags in jedem Fall funktioniert, ist daher zusätzlich die Browser-Version zu prüfen und das `HttpOnly`-Flag nur dann zu setzen, wenn der Internet Explorer ab Version 6 SP1 oder höher angetroffen wird. Da nun aber wiederum auf die Erkennung des `User-Agent`-Headers nicht immer Verlass ist, kann die Verwendung des `HttpOnly`-Flags in manchen Fällen zu Problemen mit dem Session Management führen.

Syntax: Das `HttpOnly`-Flag wird einfach als zusätzliches Attribut in den `Set-Cookie`-Header geschrieben. Beispiel:

```
Set-Cookie: SESSIONID=lk8usdjiui090iwef; path=/myApp; HttpOnly
```

Von den meisten Frameworks wird es noch nicht unterstützt. Ein Patch zur Unterstützung des `HttpOnly`-Flags in PHP4 findet sich in [6].

Siehe [7] *Protecting Data with HTTP-only Cookies*.

Bemerkung: Das `HttpOnly`-Flag verhindert nicht den schreibenden Zugriff auf ein so definiertes Cookie (auch nicht beim Internet Explorer!). Es ist daher wirkungslos gegenüber Session Fixation-Angriffen.

M170.5 RFC 2965-/ Version 1-Cookies verwenden

Wir verweisen an dieser Stelle lediglich auf das Vorhandensein einer restriktiveren Form von Cookies, den "RFC 2965"- oder auch "Version 1"-genannten Cookies. Dieses zu den herkömmlichen Cookies kompatible Format - es führt den `Set-Cookie2` Header ein - schränkt die Angreifbarkeit auf Cookies ein gutes Stück weit ein. Da sie jedoch gegenwärtig in den Browsern entweder gar nicht oder aber uneinheitlich implementiert sind, gehen wir hier nicht näher darauf ein.

Siehe [8] RFC 2965

M170.6 Session beenden

Jede Session ist aktiv zu beenden. Dazu kann die Applikation dem Benutzer die Initiative, z.B. in Form eines Logout Buttons, überlassen. Die Applikation muss aber auch eine fest eingestellte maximale Lebensdauer einer Session überprüfen. Sie darf sich auf keinen Fall auf ein bestimmtes Verhalten des Benutzers (z.B. Verwendung des Logout Button) verlassen, da dies aus vielfältigen Gründen ausbleiben kann. Folgende Kriterien sind zu beachten und an geeigneter Stelle im Programm umzusetzen:

- maximale Lebensdauer einer Session (Timeout)
- maximale Dauer, in der keine Aktion in einer Session stattfinden (Idletime)
- Logout durch Benutzer
- Logout durch Fehlersituationen

Die Sessiondaten müssen im Programm vollständig gelöscht werden, um zu verhindern, dass die Session wieder- bzw. weiter verwendet werden kann. In Java:
`session.invalidate();`

Im Browser des Benutzers sollte ein möglicherweise vorhandenes SessionID-Cookie gelöscht werden. Das kann dadurch erreicht werden, dass das Cookie erneut gesetzt wird, allerdings mit einem in der Vergangenheit liegenden `Expires`-Attribut, am besten mit dem hier gezeigten Wert (Response-Header):

```
Set-Cookie: SESSIONID=123456789; expires=Thu, 01-Jan-70 00:00:01 GMT
```

Für PHP siehe **M170.7** und **M390**.

M170.7 Handhabung von Sessions in PHP

`Session erzeugen`

Die API von PHP suggeriert eine sehr einfache Handhabung der Session-Objekte und SessionID. Leider ist diese in der Literatur verbreitete Vorgehensweise nicht im-

mer ausreichend sicher. PHP unterscheidet nicht selbst, ob eine neue Session erforderlich ist oder eine bekannte Session benutzt wird. Um wirklich neue Session-Objekte und SessionIDs zu erzeugen, ist wie folgt vorzugehen:

```
session_name('eigene-SessionID');
session_id('');
session_start();
session_id('eigener zufälliger Wert');
```

PHP verwaltet die Sessiondaten standardmäßig in Dateien. Damit stellt sich die Frage des Zugriffsschutzes auf diese Dateien. Eine alternative Methode besteht darin, für die Speicherung der Sessiondaten eine Datenbank zu benutzen. Leider bringt PHP (bis einschl. 4.x) keine Zugriffsfunktionen mit, so dass sie selbst programmiert werden müssen. In der Konfiguration wird dann der PHP-interne Handler abgeschaltet und auf den benutzerspezifischen gesetzt. In einem entsprechenden PHP-Skript müssen dann mit `session_set_save_handler()` die eigenen Funktionen registriert werden. Um dies zu ermöglichen, wird die php.ini wie folgt konfiguriert:

```
[sessions]
  session.save_handler = user
```

SessionID erneuern

In PHP kann mit `session_regenerate_id()` eine neue SessionID für eine existierende Session vergeben werden. Dieser Schritt ist auszuführen nach Authentisierung einer bereits zuvor vergebenen Session.

Session beenden

In PHP kann ein Session-Objekt mit den Funktionen `session_unset()` und `session_destroy()` zerstört werden, die Daten existieren im zugehörigen Skript aber bis zu dessen Ende weiter. Die Daten müssen explizit gelöscht werden.

Weitere Hinweise zum Umgang mit Sessions und der Konfiguration von PHP finden sich in **M390**, Abschnitt (5) "Sessions absichern"

M170.8 SessionIDs und Transportmedium

Abhängig davon, ob URL-Rewriting eingesetzt wird oder ob Cookies als Transportmittel für die SessionID verwendet werden, sind unterschiedliche Sicherheitsmaßnahmen zu ergreifen. Nur derjenige Transportmechanismus darf Verwendung finden, für den die zugehörigen Sicherheitsmaßnahmen auch umgesetzt sind. Nutzt eine Anwendung den vom Applicationserver oder der Servlet-Engine bereitgestellten Automatismus, je nach Browser-Fähigkeiten eigenständig zwischen beiden Verfahren umzuschalten, so muss die Anwendung auch für beide Verfahren die Sicherheitsvoraussetzungen erfüllen. Ist das nicht der Fall, darf der Automatismus nicht eingeschaltet sein. Im Sicherheitsprozess ist dafür Sorge zu tragen, dass eine entsprechende Abstimmung zwischen Anwendung und Konfiguration des Applicationservers hinsichtlich der SessionID-Transportart erfolgt.

M170.9 URL-Rewriting: Benutzer informieren

Erfordert die Anwendung URL-Rewriting, so sollte ein Benutzer auf die Gefahren des Ausspähens hingewiesen und dazu aufgefordert werden, sich beim Verlassen des PCs aus der Webanwendung auszuloggen oder den Rechner zu sperren. Durch Ver-

wendung sehr langer SessionIDs, die ein Abschreiben erschweren oder die Verwendung langer URLs mit der SessionID am Ende (so dass sie aus dem sichtbaren Bereich am Bildschirm herausgeschoben ist) kann zudem etwas gegen das Über-die-Schulter-schauen getan werden.

Insbesondere ist ein Benutzer darauf hinzuweisen, niemals eine gespeicherte Seite oder einen kopierten Ausschnitt aus einer solchen Seite zu versenden.

M170.10 URL-Rewriting: Externe Links dürfen keine SessionID enthalten

Es ist sicherzustellen, dass beim URL-Rewriting nicht versehentlich auch externe Links mit der SessionID versehen werden. Das nächste Beispiel weist daher keine SessionID im dritten Link auf:

```
<html><body>
...
Guten Tag Herr Meier,<br>
Sie sind eingeloggt im E-Business Portal der XYZ AG.<P>
<a href="portal?doFirma;iew98z94t30dsfn89wfe">
                          Ihre Firmendaten</a><br>
<a href="portal?doTrans;iew98z94t30dsfn89wfe">
                          Ihre Transaktionen</a><br>
<a href="https://portal.partner.tld/uid=meier">
                          Ihr Account bei unserem Partner</a><br>
...
</body></html>
```

M170.11 URL-Rewriting: Referrer-Leck verhindern

Der Referrer, der beim Einsatz des URL-Rewritings auch die SessionID enthält, darf nicht zu fremden Hosts gelangen. Dies ist dadurch sicherzustellen, dass Links auf externe Seiten niemals direkt gesetzt werden, sondern immer über einen Redirect geführt werden.

Statt also den Link zur externen Site `www.externesite.tld`, der sich auf dieser Seite befindet

```
https://www.meineanwendung.tld/getUSerinfo;JSESSIONID=koiasd756dbhvf
```

so zu setzen:

```
<a href="https://www.externesite.tld/schaumalhier">Klick</a>
```

ist dieser so zu formulieren:

```
<a href="https://www.meineanwendung.tld/redirect?url=

       https://www.externesite.tld/schaumalhier">Klick</a>
```

Nach dem Redirect ist der Referrer `https://www.meineanwendung.tld/getUSerinfo;JSESSIONID=koiasd756dbhvf` nun ersetzt durch den Redirect-Link, der die SessionID nicht mehr enthält. (vgl. auch **M160**)

Besonders dann, wenn der auf einer Seite innerhalb einer Session angezeigte Inhalt von externer Quelle stammt, ist Vorsicht geboten (Beispiel: Der Benutzer einer Shopping-Anwendung hat diesen Eintrag eingegeben, beispielsweise in einem Kommen-

tar-Feld, den der ausführende Bearbeiter sich nun ansieht). Hier kann das Referrer-Leck nicht so einfach verhindert werden wie in dem gerade beschriebenen Fall, bei dem der Inhalt unter der vollen Kontrolle des Betreibers steht. Klickt der Shop-Bearbeiter auf den enthaltenen Link, ist seine SessionID über den Referrer schon zum Angreifer übertragen. Folgende Maßnahmen sind in diesen Fällen zu ergreifen:

- Benutzereingaben sind nicht nur auf potentiell problematische HTML-Tags zu filtern. Auch `HREF`-, und sonstige Tags, die zum Ausliefern des Referrers führen, sind zu filtern.

- Die Inhalte sind in einem `INPUT`-Feld anzuzeigen (wobei selbstverständlich XSS verhindert werden muss).

Kann auf das Anzeigen solcher Links nicht verzichtet werden (Webmail-Anbieter stehen beispielsweise vor diesem Problem), so ist dies auf einer Seite durchzuführen, in der die SessionID in der URL nicht enthalten ist. Dazu ist die Anzeige der Seite wie oben gezeigt über einen Redirect zu führen. Die Session-Information ist auf einer solchen Seite dann allerdings nicht mehr vorhanden, d.h. diese Seite kann keine Links zum Fortsetzen der Session enthalten.

2.9 M180 Session Management: Bindung der IP-Adresse an die Session

Abstract **Eine Session kann durch Einbeziehung der IP-Adresse des Clients als kennzeichnendes Merkmal zusätzlich geschützt werden, sofern die daraus resultierenden Probleme in Kauf genommen werden können.**

Ebene **Implementierung**

Beschreibung

Die IP-Adresse des Client-Computers ist ein weiteres Kriterium, das zur Verhinderung von Session Hijacking herangezogen werden kann: Die Anwendung hinterlegt bei der Instanziierung der Session die IP-Adresse des Clients und überprüft bei jedem Zugriff, ob diese mit der hinterlegten IP-Adresse weiterhin übereinstimmt. Ist das nicht der Fall, wird die Session invalidiert.

Nachteilig an diesem Lösungsansatz sind deutliche Beschränkungen in Bezug auf mögliche Einsatzumgebungen: Die Bindung der IP-Adresse an die Session funktioniert nicht oder nur eingeschränkt für all jene Benutzer, die sich hinter einem Proxy befinden. Das trifft z.B. auf größere Organisationen oder auch auf Kunden eines Internet-Access-Providers zu, der eine Proxy-Architektur einsetzt. In einer solchen Umgebung kann es vorkommen, dass im Verlaufe einer Applikations-Session der ausgehende Proxy oder Router wechselt, und damit bei der Webanwendung eine andere IP-Adresse registriert wird.

Als weitere Hindernisse sind software- und netzwerktechnische Gegebenheiten zu nennen, die dazu führen können, dass bei der Webanwendung die IP-Adresse des zugreifenden Benutzers gar nicht ankommt, sondern nur diejenige eines vorgeschalteten Reverse-Proxy oder einer anderen Komponente im eigenen Netz.

Eingesetzt werden kann der vorgeschlagene Lösungsansatz jedoch in jenen Anwendungssituationen, in denen die oben genannten Nachteile nicht eintreten oder toleriert werden können. So z.B. sind die beschriebenen Hindernisse in der Regel in Intranet-Bereichen nicht vorhanden, so dass hier die Bindung der Session an die IP-Adresse bei Anwendungen mit erhöhtem Schutzbedarf als eine zusätzliche Sicherheitsmaßnahme erfolgen kann.

2.10 M190 Session Management: Sicherheit erhöhen durch zusätzliche Parameter

Abstract **Die Session ist durch Verwendung weiterer identifizierender Merkmale des Clients zusätzlich zu schützen.**

Ebene **Implementierung**

Beschreibung

Problematisch für das Session-Management ist die Übertragung der SessionID über einen einzelnen Träger. Der Gedanke liegt daher nahe, die SessionID auf verschiedene Träger zu verteilen, und so den gleichzeitigen Zugriff auf mehrere Träger zu erschweren oder gar zu verhindern - oder aber weitere Merkmale an die Identifikation der Session zu binden.

`User-Agent-Header`

Im `User-Agent`-Header schickt der Browser seine Typenbezeichnung mit. Die Anwendung könnte diese bei der Instanziierung der Session darin hinterlegen und bei jedem Zugriff prüfen, ob der `User-Agent`-Header mit dem hinterlegten weiterhin übereinstimmt. Ist das nicht der Fall, wird die Session invalidiert.

Dieses Vorgehen bringt keine oder nur unwesentliche Verbesserungen gegenüber:

- Sniffing, Ausspähen, Referrer-Sniffing, Scripting

Eine Verbesserung wird erreicht gegenüber:

- Speichern, Copy/Paste
- Session Fixation: Ein Angriff kann dadurch nicht verhindert werden. Hierzu müsste zunächst die Browserversion des Benutzers ermittelt werden.

Bewertung: Auch wenn diese Maßnahme keinen wirklichen Schutz darstellt, ist sie doch zu empfehlen. Sie ist recht leicht durchführbar und hat außer dem zusätzlichen Implementierungsaufwand keine weiteren Nachteile.

`Accept- und Accept-Language-Header`

Diese Header weisen nur eine sehr geringe Variationsbreite auf. Es gelten ansonsten dieselben Einschränkungen wie beim `User-Agent`-Header. Sie sind als zusätzliches Identifikationsmerkmal daher auch nur bedingt geeignet.

`Referer-Header`

Der `Referer-Header` enthält die URL der zuletzt besuchten Seite (genauer eigentlich: der Seite, in der sich der geklickte Link befindet). Mit seiner Hilfe sind ausgeklügelte und aufwändige Verfahren für ein sehr sicheres Session Management implementierbar. Siehe dazu z.B. [9].

Im Sinne einer einfachen Lösung kann der `Referer`-Header ähnlich wie der `User-Agent`-Header verwendet werden. Die Anwendung würde dann bei jedem Zugriff zu-

sätzlich prüfen, ob der Referrer auch mit einem für alle Zugriffe festen Teil des Pfades beginnt, also beispielsweise: `https://www.test.tld/meineAnwendung/`.

Es ist zu berücksichtigen, dass einige Browser erlauben, die Übermittlung des Referrers zu deaktivieren. Auch Content-Filter (sowohl im Browser als auch auf Proxys) können entsprechend eingestellt werden.

Bemerkung

Ein Problem aller beschriebenen Ansätze stellt die Tatsache dar, dass diese von den vorhandenen APIs – namentlich der Servlet-API – nicht unterstützt werden und von der Anwendung selbst implementiert werden müssten.

2.11 M195 Session Management: Kombination verschiedener Träger mit Dialogtracking

Abstract **Hier werden die Verfahren zur Implementierung erhöhter Sicherheit beim Session Management genannt.**

Ebene **Implementierung**

Beschreibung

Der Weg zu einem Session Management mit besseren Sicherheitseigenschaften führt über eine Art Kombination der Cookie-Technik mit der Technik des URL-Rewritings.

`Variante 1`

Zusätzlich zur SessionID (die im Cookie transportiert wird) wird mit der Erzeugung der Session eine weitere, von der SessionID verschiedene, nicht erratbare ID erzeugt, die wir *SessionCode* nennen. Der SessionCode wird in jeder URL als zusätzlicher Parameter eingefügt. Beim Aufruf der URL (d.h. Klick auf den Link oder Button) prüft die Anwendung die Übereinstimmung des übergebenen mit dem in der Session hinterlegten SessionCode. Stimmen sie nicht überein, so wird die Ausführung gestoppt und die Session invalidiert.

Dieses Verfahren ist sehr ähnlich dem in **M220** beschriebenen Verfahren zur Verhinderung von Session Riding. Weitere Details sowie ein Implementierungsbeispiel in Java können dieser Maßnahme entnommen werden.

Das Verfahren vereint die jeweiligen Vorteile von Cookies und URL-Rewriting. Wird das Verfahren wie in **M220** beschrieben implementiert, so stellt es gleichzeitig eine wirksame Maßnahme gegen Session Riding dar.

`Variante 2`

Die sicherste Form des Session Managements erreicht man durch ein Dialogtracking. Dabei ist der oben eingeführte SessionCode nicht statisch, sondern wird bei jedem Zugriff neu vergeben. Wir bezeichnen jenen SessionCode als *Token*.

Die Anwendung baut in jeden Dialogschritt (d.h. jeden Link in einer Seite) das diesen Dialogschritt eindeutig kennzeichnende *Token* ein. Erhält sie einen Request, prüft sie, ob das Token für diesen Aufruf gültig ist und streicht es aus der Liste der gültigen Token. Ein erneuter Aufruf dieses Links (Replay durch den Angreifer) ist damit nicht mehr möglich. Auch ist es nicht möglich, eine Aktion ohne Token aufzurufen. Die Kenntnis der SessionID und eines auf dem Übertragungsweg ausgespähten Tokens ist damit nicht mehr ausreichend, um in die Session einzudringen. Man muss gleichzeitig auch das Token von mindestens einem "unverbrauchten" (d.h. noch nicht aufgerufenen) Link besitzen.

2.12 M200 Session Management: Logout erzwingen

Abstract **Jede Anwendung, die eine Login-Funktion hat, sollte auch eine Logout-Funktion besitzen. Das Logout hat das korrekte Invalidieren der Session sicherzustellen.**

Ebene **Logik**

Beschreibung

Das Logout muss die SessionID zuverlässig invalidieren. Bei zahlreichen Webanwendungen ist zu beobachten, dass eine Logout-Funktion zwar vorhanden ist und der Benutzer nach Ausführung auch die Bestätigung erhält, nun abgemeldet zu sein, dass die Session aber nicht invalidiert wird. Zu erkennen ist das meist daran, dass die Betätigung des Zurück-Buttons des Browsers bis zu einer Seite innerhalb der Anwendung ein Weiterarbeiten darin nach wie vor ermöglicht. Mit einer gestohlenen SessionID könnte in einem solchen Fall weitergearbeitet werden (Stichwort "Internet-Café").

Der Benutzer ist darin zu trainieren, das Logout auch wirklich auszuführen. Dazu ist

- der Logout-Button deutlich sichtbar darzustellen.
- der Benutzer nach dem Login darauf hinzuweisen, sich nach Beendigung auch wieder auszuloggen.
- der Benutzer darauf hinzuweisen, sich beim nächsten Mal korrekt auszuloggen, falls dies beim letzten Mal nicht geschehen ist.

Schließt der Benutzer das Browser-Fenster in der Annahme, sich dadurch auch automatisch auszuloggen (was ja nicht der Falls ist), so sollte die Anwendung den Benutzer beim nächsten Login darüber informieren, dass ein automatisches Ausloggen erfolgte, und in Zukunft ein explizites Ausloggen stattfinden sollte.

2.13 M210 Anforderungen an eine SessionID

Abstract **Wird die Erzeugung der SessionID nicht einer ausgereiften API überlassen, sondern selbst implementiert, sind bestimmte Anforderungen zu beachten.**

Ebene **Logik, Implementierung**

Beschreibung

Nicht immer kann oder soll die Erzeugung und Verwaltung einer SessionID von einer ausgereiften API (Servlet-Engine, Applicationserver, ...) ausgeführt werden. Für diese Fälle sollten folgende Anforderungen berücksichtigt werden:

- Keine Verknüpfung mit extern bekannten oder erratbaren Daten. Dazu gehören: IP-Adresse des Clients, Uhrzeit, Prozess-ID, Attribute des Benutzers wie Geburtsdatum oder externe Schlüssel wie die E-Mail-Adresse.
- Hohe Zufälligkeit (Randomness): Das Erzeugungsmuster darf nicht aus einer Menge von Proben ableitbar sein.
- Die SessionID muss eine ausreichende Länge haben, um gegenüber Brute-Force-Angriffen – dem Durchprobieren aller Möglichkeiten – zu bestehen.
- Sie sollte ausschließlich über sichere Kanäle übertragen werden, wenn der Schutzbedarf der Anwendung dies erfordert. Siehe **M170.8** und **M310**.
- Die SessionID einer Anwendung, deren Schutzbedarf SSL erfordert, darf niemals versehentlich über eine unverschlüsselte Verbindung mit übertragen werden. Siehe **M170.3** (secure-Flag setzen).
- Die Session selbst muss das kürzestmögliche Ablaufdatum haben, das für den jeweiligen Anwendungszweck gerade noch angemessen ist.

2.14 M220 Session Riding

Abstract **Session Riding ist durch Einführung eines SessionCodes oder durch eine andere Schutzmaßnahme zu verhindern.**

Ebene **Implementierung**

Beschreibung

Die im Folgenden angeführten Maßnahmen sind geeignet, um Session Riding zu ver-

Schwachstellen, so können diese jedoch eine Schutzmaßnahme wieder aushebeln.

`Einfügen eines SessionCodes`

Eine Anwendung ist immer dann anfällig für Session Riding-Angriffe, wenn der Request mitsamt all seinen GET- und POST-Parametern erratbar ist. Daraus folgt bereits, dass Anwendungen, welche die SessionID mit der Technik des URL-Rewritings transportieren, nicht verwundbar sind, denn einer der Parameter ist gerade die nicht erratbare SessionID.

Die einfachste Schutzmaßnahme für Anwendungen, die die SessionID per Cookie transportieren oder eine schwache Authentisierungsmethode verwenden, beruht nun darauf, einen geheimen Schlüssel (*SessionCode* – siehe **M195**) als zusätzlichen Parameter einzuführen. Der SessionCode ist bei Eintritt in die Session zu vergeben und wenigstens in alle Zugriffe einzubauen, die zu Zustandsänderungen führen oder Seiteneffekte auslösen können. Er ist entweder in die URL zu schreiben oder als `Hidden`-Field zu übertragen. Der SessionCode darf nicht als Cookie gesetzt werden, da er dann genauso wie die SessionID automatisch mitgeschickt würde. Die Anwendung überprüft bei jedem solchen Aufruf, ob der SessionCode mit dem in der Session hinterlegten übereinstimmt und lehnt im Fehlerfall die Ausführung ab.

Es sei darauf hingewiesen, dass die Verhinderung von Session Riding gegenwärtig von den vorhandenen APIs zum Session Management nicht unterstützt wird. Der Aufwand, die beschriebene Maßnahme für bestehende Anwendungen umzusetzen, kann daher beträchtlich sein. Weiter unten geben wir ein Codebeispiel in Java

Diese Maßnahme ist dann nicht erforderlich, wenn alle zustandsändernden Requests vor der Ausführung vom Benutzer mit der Eingabe eines nicht erratbaren Wertes bestätigt werden. Dieser Fall kann beispielsweise bei einer Online-Anwendung vorliegen, die für derartige Abfragen die Eingabe eines Einmal-Passwortes verlangt.

Der *SessionCode* kann auch die SessionID selbst sein. Dann ist jedoch die Gefahr gegeben, dass durch Ausnutzung einer möglicherweise vorhandenen XSS-Schwachstelle das Cookie, und damit auch der SessionCode, gestohlen werden kann. Wenn es keinen höheren Aufwand bedeutet, für den SessionCode eine von der SessionID verschiedene ID zu verwenden, ist dies vorzuziehen. Damit ist dann auch gleich eine deutlich höhere Sicherheit vor Session Hijacking gegeben.

Umsetzung in Java

Nach dem erfolgreichen Login des Benutzers wird der (hinreichend lange und hinreichend zufällige) SessionCode `UniqueLoginSessionCode` erzeugt und in der Session abgelegt. Das geschieht gleich nach den Vorkehrungen zur Verhinderung von Sessi-

on Fixation (**M170.1**):

```
public void setAuthenticatedUser(UserIF userObject)
{
  protectSessionFixation();
  getSession().setAttribute(ControllerConstants.SESSION_USER,
                                                        userObject);
  getSession().setAttribute(LOGIN_SESSIONCODE,
                 ControllerUtils.getNextUniqueLoginSessionCode());
}
```

Der SessionCode ist, wenn nicht grundsätzlich, so doch zumindest in allen Fällen, wo zustandsändernde Operationen ausgeführt werden, als GET-Parameter in den Link oder `Hidden`-Field in die Form zu schreiben. In der Controller-Schicht der Anwendung ist dann der erhaltene Wert mit dem in der Session gespeicherten zu vergleichen. Stimmen sie nicht überein oder fehlt der Parameter, so ist die Verarbeitung abzubrechen und die Session sicherheitshalber zu invalidieren.

Abfrage eines SessionCodes

In speziellen Fällen kann eine weitere Maßnahme sinnvoll sein, die jedoch aus Sicht des Benutzers mit wiederum erhöhtem Aufwand verbunden ist: Nach dem Absenden eines kritischen Requests durch einen Benutzer wird zunächst eine Seite mit einer zufälligen Zeichenfolge (eine andere Form eines SessionCodes) angezeigt. Erst wenn diese Zeichenfolge durch Eintippen in ein entsprechendes Eingabefeld, und nachfolgendes Übersenden an die Webanwendung bestätigt wird, erfolgt die Ausführung des kritischen Requests. Auch der Einsatz von Captchas kommt hierfür in Frage (**M270.1**).

Referrer-Test

Eine wichtige Regel lautet, sich nicht alleine auf den `Referer`-Header zu verlassen. In diesem Fall können wir eine Ausnahme machen und ihn als zusätzliches Merkmal, welches dem Benutzer nicht von außen untergeschoben werden kann, verwenden (von den unten genannten Einschränkungen einmal abgesehen). Diesem Ansatz liegt die Annahme zugrunde, dass ein Request nur dann legitim ist, wenn er durch einen Klick auf einer zur Anwendung gehörenden Seite ausgelöst worden ist. Und immer dann ist auch der `Referer`-Header mit der URL dieser Seite belegt. Die Anwendung muss also prüfen, ob der übergebene Referrer in der Liste der in Frage kommenden URLs vorhanden ist und kann im Positiv-Fall davon ausgehen, dass es sich um die legitime Verwendung handelt. Wie der nächste Absatz zeigt, ist es wichtig, dabei die gesamte URL zu matchen und nicht nur den Host-Anteil.

Einschränkung 1

Der so erreichte Schutz kann durch eine XSS-Schwachstelle auf derselben Site – ggf. auch außerhalb der Anwendung selbst – wieder zunichte gemacht werden. Zwar lässt sich der Referrer per JavaScript nicht direkt manipulieren, über den Umweg des Cross-Site Scripting ist es dann aber doch möglich.

Einschränkung 2

Der Referrer ist nicht immer verfügbar, da ihn manche Content-Filter auf Client-Seite herausfiltern.

Bemerkung

Wird eines der beiden in **M195** beschriebenen Verfahren zur Verhinderung von Session Hijacking implementiert, so wird damit auch gleichzeitig Session Riding wirksam verhindert.

2.15 M225 Privilege Escalation verhindern

Abstract **Verhinderung von Privilege Escalation in modernen Webanwendungen.**

Ebene **Implementierung**

Beschreibung

Privilege Escalation bezeichnet die unautorisierte Erhöhung der Privilegien, die einem angemeldeten Benutzer zugeordnet sind.

Beispielhaft habe eine Anwendung zur Bestellabwicklung für jede neue Bestellung eine eindeutige ID vergeben. Diese IDs werden in der Reihenfolge der Bestellaufgabe und über alle Benutzer hinweg sequentiell erzeugt. Ein Benutzer kann sich die eigenen Bestellungen über eine Web-Schnittstelle anzeigen lassen. Ausgegeben wird eine Liste von Bestellungen, die durch eine Folge nun nicht mehr zusammenhängender IDs identifiziert werden. Nur auf diese Bestellungen soll der Benutzer zugreifen dürfen. Beim anschließenden weiteren Zugriff auf eine konkrete Bestellung wird die ID als Parameter mit übergeben. *Privilege Escalation* kann nun dann vorliegen, wenn die Anwendung beim Zugriff auf einen Datensatz nicht mehr prüft, ob der Benutzer rechtmäßiger Eigentümer dieses Datensatzes ist, sondern einfach die als Parameter übermittelte ID zum Zugriff auf die Datenbank heranzieht.

Zur Verhinderung von Privilege Escalation muss beim Bearbeiten oder Einsehen eine Bestellung kontrolliert werden, ob die über die ID identifizierte Bestellung entweder dem anfragenden Benutzer gehört oder ob der Benutzer eventuell der Administrator ist.

In einer modernen Anwendung mit einer Model-View-Controller (MVC)-Struktur ist dies relativ einfach zu bewerkstelligen, indem die Kontrollfunktion im Modell (der Business-Logik) verankert wird:

```
public void updateOffer(Offer offerToBeUpdated, User callingUser)
throws FakedIdException
{
  if (!hasUserRightToUpdate(offerToBeUpdated, callingUser))
     throw FakedIdException("updateOffer: "
            + offerToBeUpdated + " - " + callingUser);
  // updateOffer...
}
```

Durch das Wirksamwerden der Exception wird sofort an allen Stellen im Code angezeigt, wo eine Reaktion erfolgen muss.

Die Praxis zeigt jedoch, dass diese Kontrolle häufig ganz oder teilweise in der Controller-Schicht vorgenommen wird. Nachteile sind duplizierter Code und dass ein Entwickler vergessen könnte, die Kontrolle überhaupt einzubauen. Ein Grund dafür ist, dass an Sicherheit meist zu einem sehr späten Zeitpunkt gedacht wird und Änderungen am Business-Modell dann zu aufwändig sind. Ein anderer Grund sind funktionale Erweiterungen wie z.B. die nachträgliche Einführung des oben genannten Administrators, die eine grundsätzliche Überarbeitung des Sicherheitskonzepts notwendig machen würde.

2.16 M230 POST erzwingen

Abstract **POST-Requests sind GET-Requests vorzuziehen oder zu erzwingen.**

Ebene **Implementierung**

Bezug **allgemein**

Beschreibung

Die Ausnutzung einer XSS-Lücke zum Zwecke des Website-Spoofings ist dann am einfachsten durchführbar, wenn der manipulierende Code einfach an den Aufruf angehängt werden kann. Ein Session Riding-Angiff lässt sich dann in einem IMG-Tag verstecken und unbemerkt ausführen. Ähnliches gilt für andere Angriffsformen.

Möglich ist dies immer dann, wenn der Request von der Anwendung als GET-Request behandelt wird. Verwendet die Anwendung an dieser Stelle jedoch die POST-Methode, ist ein Einbau in die URL nicht mehr möglich.

Es empfiehlt sich daher, HTML-Forms per POST-Methode zu übertragen, so dass im Falle einer vorhandenen XSS-Lücke wenigstens eine kleine zusätzliche Hürde gesetzt wird. Allerdings abstrahieren viele heutige Frameworks und Komfortfunktionen über der Request-Methode, d.h. sie liefern die übergebenen Parameter, egal ob diese per GET oder POST geschickt worden sind. Selbst dann, wenn in der Form ein POST als Request-Methode angegeben ist, kann ein Angreifer die Daten an die URL hängen und per GET übertragen, da die Anwendung nicht unterscheidet. Der Ausschluss von GET ist daher von der Anwendung zu erzwingen.

Beispiele

Beispiel in Java

POST-Parameter können in Java Servlets z.B. auf folgende Weise gelesen werden:

```
protected void doPost(HttpServletRequest request,
  HttpServletResponse response)

    throws ServletException, IOException

{  ...

    if ( request.getMethod().toLowerCase().equals("get") )

    {

      // request abweisen

    }

    String param1 = request.getParameter("param1");

}
```

Beispiel in PHP

Statt von globalen Variablen `$_GET` wird von `$_POST`, aber niemals von `$_REQUEST` gelesen.

```
$clean = $_POST['par'];
```

`GET`-Requests werden hier nicht angenommen.

Bemerkung

Die ursprüngliche Idee hinter `GET` und `POST` (siehe [10]) war, zwischen *sicheren* und *praktischen* Anforderungen zu unterscheiden. Es muss festgestellt werden, dass die Anforderungen der Praxis und die hier beschriebenen Sicherheitsgesichtspunkte nicht mehr mit dieser Idee in Einklang zu bringen sind. Heute muss *sicher* nicht nur im Kontext des Benutzers, sondern auch im Kontext der Anwendung betrachtet werden. Das *praktische* `GET` muss dann durch das etwas *sicherere* `POST` ersetzt werden.

2.17 M250 Minimalitätsprinzip für Informationen

Abstract **Bei der Bereitstellung von Informationen ist abzuwägen zwischen der bestmöglichen Unterstützung des Benutzers und dem Schutz vor Angriffen.**

Ebene **Semantik**

Beschreibung

In der Regel ist es empfehlenswert, nur diejenigen Informationen bereitzustellen, die für die Anwendung notwendig sind. Darüber hinaus gehende Informationen könnten unnötige Ansatzpunkte für eine Kompromittierung der Webanwendung liefern.

Siehe auch **M320** "Information Disclosure verhindern"

Beispiele

Falsch: "Bitte geben Sie ihre Benutzerkennung und die 6-stellige PIN ein"

Richtig: "Bitte geben Sie ihre Benutzerkennung und ihre PIN ein."

Falsch: "Das eingegebene Passwort ist falsch", falls die Benutzerkennung richtig und das Passwort falsch eingegeben worden sind

Richtig: "Login nicht möglich. Benutzerkennung oder Passwort nicht korrekt."

Falsch: "Der Benutzer existiert nicht", falls die Benutzerkennung nicht existiert.

Richtig: "Login nicht möglich. Benutzerkennung oder Passwort nicht korrekt."

Falsch: Eintrag auf der Hilfe-Seite: "Geben Sie hier Ihre Kundennummer ein. Sie finden die Kundennummer rechts oben auf jeder Rechnung. Es ist die 10-stellige Zahlenfolge, die mit einem R beginnt. "

Richtig: Verzicht auf die exakten Angaben über Formate, Längen, Datentypen und Informationen, aus denen auch ein Außenstehender Hinweise für deren Erlangung entnehmen kann. Stattdessen "... in der Form, wie wir es Ihnen in unserem Anschreiben mitgeteilt haben."

Der Informationsgehalt von Demoanwendungen, die öffentlich zugänglich sind, ist unter Sicherheitsgesichtspunkten zu beurteilen und möglichst einzuschränken.

Auf keinem Fall sind Demoanwendungen als Instanzen der Produktivanwendung bereitzustellen.

Ausführliche Erklärungen sind nur dem an der Anwendung angemeldeten Benutzer zu geben.

M250.2 Hilfe-Seiten nur dem angemeldeten Benutzer zeigen

Hilfe-Seiten, die Informationen über Zusammenhänge von geschützten Anwendungen enthalten, dürfen auch nur im geschützten Bereich zugreifbar sein.

2.18 M260 Identitätsprinzip

Abstract

Ebene **Semantik**

Beschreibung

Ein Benutzer ist in die Lage zu versetzen, Manipulationen, Fälschungen und Täuschungen zu erkennen. Dazu muss aus dem Umgang mit der Website, aber auch aus der gesamten Kommunikation mit dem Unternehmen erkennbar sein, dass das Unternehmen bestimmte Konventionen durchgängig einhält. Zeigt sich im Kontext einer Webanwendung ein auffälliges, nicht den Konventionen entsprechendes Verhalten, wird ein Benutzer automatisch misstrauisch und verhält sich zurückhaltender, so dass es einem Betrüger deutlich schwerer fällt, sein Ziel zu erreichen. Diese Konventionen sollten den Benutzern darüber hinaus explizit mitgeteilt werden. Im Weiteren sollten Verhaltenshinweise für den Fall der Verletzung der Konventionen genannt werden.

Das Prinzip, welches auch der Corporate Identity eines Unternehmens zugrunde liegt, nämlich die Schaffung einer unverwechselbaren Identität und eines Wiedererkennungseffektes (Content Branding), wird mit dieser Maßnahme auch auf die Online-Kommunikation angewandt.

Beispiele

Wir nennen hier, stellvertretend für viele weitere Gesichtspunkte, einige Beispiele:

- Popup-Fenster sind unter dem Aspekt der Sicherheit sehr problematisch. Nach wie vor existiert eine Reihe von technischen Möglichkeiten (nicht nur bei Vorliegen einer XSS-Schwachstelle), ein betrügerisches Fenster so vor der Seite einer authentischen Website zu positionieren, dass es den Anschein hat, dass das Fenster zu der Website gehört. Ein Anbieter, der die Benutzer daran gewöhnt hat, dass immer wieder unvermittelt Popups erscheinen, nimmt den Benutzern die Möglichkeit, in solch einem Fall Verdacht zu schöpfen. Popups sind daher sehr sorgfältig auf ihre Auswirkungen auf die Sicherheit hin zu prüfen. Im Zweifel sollte auf deren Einsatz verzichtet werden.

- Für Werbebanner gilt Ähnliches wie für Popups. Sie führen dem Benutzer immer wieder Überraschendes und nicht im Kontext der Website stehendes vor Augen und befinden sich dabei im Vertrauenskontext der Website: Das Vertrauen, welches dem Anbieter entgegen gebracht wird, überträgt sich auf den Inhalt des Werbebanners. Besteht nicht eine starke Kontrolle über die Inserenten, so kann hier ebenfalls schnell Missbrauch entstehen. Und auch in diesem Fall gilt, dass eine XSS-Lücke häufig die Möglichkeit bietet, etwas zu überlagern oder einzufügen, das dann als Werbebanner verstanden wird.

- Unerwartetes und Überraschendes sollte möglichst nicht geschehen. Dazu gehört auch, dass ein Benutzer nicht ohne besonderen Grund ausgeloggt wird bzw. erneut zum Login aufgefordert wird (ein besonderer Grund besteht, wenn der Benutzer zu lange inaktiv geblieben ist). Ein Angreifer hätte es dann leichter, einem Benutzer bei Bestehen einer entsprechenden Sicherheitslücke eine gefälschte Login-Seite unterzuschieben.

- Werden an die Benutzer E-Mails geschickt, dann sollten diese immer von ein und derselben E-Mail-Adresse kommen, z.B. `service@meinefirma.de`, niemals von `noreply@maildienstleister.tld`[3]

- Ein SSL-Serverzertifikat sollte niemals fehlerhaft sein. Immer wieder trifft man auf den Fall, dass der Servername, wie er in der URL steht, nicht exakt mit dem im Zertifikat eingetragenen Namen übereinstimmt. Dies führt zu einer Warnmeldung des Browsers, die geeignet ist, die Benutzer nicht nur zu verunsichern, sondern auch dazu führen kann, dass Derartiges zukünftig als normal hingenommen wird. Ebenso sollte das Überschreiten der Gültigkeitsdauer eines Zertifikats vermieden werden.

[3] Wegen der leichten Fälschbarkeit von E-Mail-Adressen würde auch ein Angreifer diese Absendeadresse benutzen. Trotzdem ist es im Sinne dieser Maßnahme wichtig, auf Durchgängigkeit der Konventionen zu achten.

2.19 M270 Verhinderung von Überflutung / Denial-of-Service durch automatisierte Angriffe

Abstract **Es ist sicherzustellen, dass Ressourcen von unberechtigten Dritten nicht blockiert werden können.**

Ebene **Logik**

Beschreibung

Eingangsbemerkung: Diese Maßnahme konkurriert mit Maßnahme **M280.**

Grundlegendes Problem jeder Maßnahme zur Verhinderung von DoS-Angriffen ist die Unterscheidung eines solchen Angriffs von legitimen Zugriffen, sowie die gezielte Blockierung des Angreifers, ohne legitime Benutzer in Mitleidenschaft zu ziehen. Wir gehen bei dieser Maßnahme davon aus, dass ein Angreifer eine Überflutung auf der logischen Ebene nicht durch manuelle Browserinteraktionen ausführen kann, sondern dazu die Zugriffe automatisieren, d.h. sich eines geeigneten Skripts oder Tools bedienen muss. Unter dieser Annahme reduziert sich die Aufgabe darauf, automatisierte Zugriffe zu blocken und manuelle Zugriffe zuzulassen.

Ein automatisierter DoS-Angriff kann wirksam dadurch verhindert werden, dass der Benutzer nach Erreichen einer festgelegten Toleranzschwelle von erlaubten Wiederholungen aufgefordert wird, eine Eingabe zu tätigen, die ein Programm nicht liefern kann. Wird die Eingabe korrekt gegeben, kann davon ausgegangen werden, dass kein automatisierter Angriff stattfindet. Folgende Verfahren kommen in Frage:

M270.1 **Mustererkennung / Captchas**

Das Verfahren der *Captchas* (Completely Automated Public Turing Test to Tell Computers and Humans Apart) [11] macht sich die Überlegenheit des Menschen gegenüber der Maschine in der visuellen Mustererkennung zunutze. Dem Benutzer wird ein Bild gezeigt, welches z.B. eine mit Störungen versehene Zeichenkette enthält, die von einem Menschen gerade noch zu erkennen ist, von einer Maschine aber nicht mehr erkannt werden kann. Die korrekte Eingabe der Zeichenkette schaltet die Sperrung wieder frei.

Beim Einsatz von Captchas und der Auswahl einer Captcha-Bibliothek ist zu beachten, dass die Sicherheit dieses Verfahrens mit der maschinellen Nichtlösbarkeit der Aufgabe steht und fällt. Fortschritte in der Mustererkennung oder die Entdeckung von Algorithmen können diese Art von Schutz mit einem Schlag wertlos machen. [12] stellt eine frei verfügbare Captcha-Klasse für ASP zur Verfügung. Ein weiterer möglicher Nachteil von (schlecht gemachten oder falsch eingesetzten) Captchas ist die Behinderung der Barrierefreiheit, wenn z.B. das Entziffern die Wahrnehmung von Farben (Farbenblindheit!) voraussetzt.

Beispiel

Eine Website stellt ein Kontaktformular bereit. Dieses wird nach Ausfüllen und Absenden durch einen Benutzer per E-Mail an die dafür vorgesehene Kontaktperson gesandt. Um zu verhindern, dass ein Angreifer mittels eines Skripts über das Kontaktformular eine große Anzahl von E-Mails generiert und die Mailbox der Kontaktper-

son überflutet, enthält das Formular ein Captcha. Der Benutzer wird aufgefordert, den Text, den er in der Captcha-Grafik erkennt, in das dafür vorgesehene Feld einzutippen. Nur wenn Übereinstimmung besteht, leitet die Anwendung die Anfrage per E-Mail weiter.

M270.2 Rätselfrage

Dem Benutzer wird eine Frage gestellt, die eine Maschine nicht interpretieren und damit nicht beantworten kann. Beispiele:

"Wie lautet der dritte Buchstabe im fünften Wort dieses Satzes? "

"Geben Sie das Wort Autobaan ohne Rechtschreibfehler ein! "

"Aus wie vielen Buchstaben besteht das erste Wort des nächsten Satzes? "

Problem bei diesem Ansatz ist die Schwierigkeit, eine genügend große Anzahl von Fragen zu generieren. Ist die Anzahl nicht groß genug, kann ein Angreifer sämtliche Fragen abfragen, manuell die Antworten erstellen und die Zuordnung von beidem dem Automaten geben, der dann anhand des Fingerprints zu jeder Frage die richtige Antwort findet.

Beispiel

Eine Zeitung bietet im öffentlichen Bereich ihres Webauftritts eine Volltextsuche über alle jemals erschienenen Artikeln an. Da jede Suchanfrage das System merklich belastet und bereits bei rund 20 gleichzeitigen Anfragen eine deutliche Verschlechterung der Antwortzeit zu beobachten ist, sind Maßnahmen zur Verhinderung eines DoS-Angriffs zu ergreifen. Daher wird der Benutzer im Formular für die Eingabe der Suchworte zusätzlich aufgefordert, auf eine Rätselfrage zu antworten. Da die mögliche Schadenshöhe als niedrig eingestuft wird, reicht es, eine Menge von 100 unterschiedlichen Fragen vorzubereiten und diese zufällig in die Suchanfragen einzustreuen.

2.20 M280 Enumeration verhindern

Abstract **Die Zugriffe auf Ressourcen, die durch Enumeration-Techniken verwundbar sind, sind zu monitoren und Wiederholungen angemessen zu begrenzen.**

Ebene **Logik**

Beschreibung

Ein Enumeration-Angriff muss zunächst erkannt werden. Ist er erkannt, ist der weitere Zugriff in geeigneter Weise zu blocken oder zu behindern. Vgl. auch **M270**.

2.20.1 Erkennung von Enumeration-Angriffen

Die Erkennung eines Enumeration-Angriffs bzw. die Unterscheidung von legitimen Zugriffen ist häufig nicht eindeutig zu treffen. Das im Folgenden beschriebene Verfahren gibt eine grobe Richtschnur.

Um einen Enumeration-Angriff handelt es sich, wenn der Zugriff auf eine durch Enumeration-Techniken verwundbare Ressource erfolgt und zusätzlich eines oder mehrere dieser Merkmale zutreffen:

- Die Zahl der Zugriffsversuche innerhalb einer bestimmten Zeitspanne übersteigt das übliche Maß deutlich.
- Die Zugriffsparameter werden nach einem regelmäßigen Muster variiert.
- Der Zugriff erfolgt immer von derselben Stelle (d.h. IP-Adresse).

Da das zweite Merkmal leicht verschleiert werden kann, verwenden wir es im Weiteren nicht, um Gegenmaßnahmen daraus abzuleiten.

Das dritte Merkmal kann ebenfalls verschleiert werden, nämlich dadurch, dass die Zugriffe von verschiedenen IP-Adressen aus erfolgen. Der Angreifer muss dazu die Zugriffe über unterschiedliche Proxy-Server laufen lassen. Das Rotieren über einige wenige Proxys ist dabei leicht möglich und muss von einer Abwehrmaßnahme erkannt werden. Theoretisch ist wegen der großen Zahl im Internet vorhandener Proxys auch das Rotieren über eine große Zahl von Proxys möglich. In der Praxis ist dies jedoch nur unter hohem Aufwand zu erreichen, da die meisten dieser Proxys recht unzuverlässig sind. So sind nur Angreifer mit ausgeprägten Erfahrungen oder einer eigenen Infrastruktur in der Lage, derartig vorzugehen.

Das folgende Verfahren stützt sich im Wesentlichen auf das erste der drei genannten Merkmale:

(1) Identifikation der Ressourcen, die verwundbar für Enumeration sind. Dies können z.B. sein: Benutzerkennungen, SessionIDs, Dokumentennummern, usw.

(2) Für jede verwundbare Ressource: Ermittlung einer Zahl N von Zugriffen innerhalb einer definierten Zeitspanne, die erlaubt werden kann, ohne die Ressource der Gefahr für Enumeration auszusetzen.

(3) Für jede verwundbare Ressource: Ermittlung, wieviele Zugriffe unter normalen Umständen zu Spitzenzeiten in der gewählten Zeitspanne erfolgen.

(4) Festlegung eines Mittelwertes auf Basis von (2) und (3), der die Höchstzahl von Zugriffen auf die Ressource innerhalb der gewählten Zeitspanne darstellt. Zählung diese Zugriffe, und Blockade (s.u.) aller weiteren Zugriffe.

Je nach Anwendungsfall kann das Verfahren global auf eine Ressource, also unabhängig von weiteren Parametern, angewendet werden oder aber an andere Parameter gekoppelt sein.

Beispiel 1 – Enumeration von Benutzerkennungen

Der Registrierungsprozess einer Website erlaubt einem neuen Benutzer die Angabe einer eigenen Benutzerkennung. Wählt ein Benutzer einen Namen, der bereits existiert, so muss die Anwendung die Annahme verweigern und zur Eingabe eines neuen Namens auffordern. Ein Angreifer könnte nun eine Liste von Benutzernamen dadurch erstellen, dass er die Anmeldung systematisch mit immer anderen möglichen Namen durchläuft und an der Art der Antwort erkennt, ob der Benutzername existiert oder nicht.

In Schritt 1 des obigen Verfahrens wird die Wahl des Benutzernamens im Registrierungsprozess als mögliche Problemstelle für Enumeration identifiziert.

In Schritt 2 wird ermittelt, dass ein Angreifer innerhalb einer Stunde mindestens 10.000 Versuche haben muss, um durch Enumeration zu verwertbaren Ergebnissen zu kommen.

In Schritt 3 wird festgestellt, dass die durchschnittliche Zahl von Anmeldungen ca. 1 Anmeldung pro Stunde beträgt und Spitzenwerte bei 10 Anmeldungen pro Stunde liegen. Im Schnitt erfolgen 2 Versuche, bevor ein noch freier Benutzername identifiziert wird. Der gefragte Wert errechnet sich damit auf 20 Zugriffe pro Stunde.

In Schritt 4 wird als Ergebnis dieser Analyse die Registrierungsprozedur mit einer Prüfung ausgestattet, die weitere Zugriffe blockt, wenn innerhalb von 6 Minuten mehr als 30 Zugriffe erfolgen.

Beispiel 2 – Passwort-Enumeration

Eine Webanwendung verwendet als Passwort 5-stellige numerische PINs. Werden keine Vorkehrungen gegen Enumeration getroffen, so ist bei bekanntem Benutzernamen eine PIN mit durchschnittlich 50.000 Versuchen ermittelbar.

Es liegt auf der Hand, dass bei derartig geringer Streuung ("Randomness") und gleichzeitig hohem Schutzbedarf der Spielraum für Fehlversuche sehr eng zu halten ist, d.h. ein Blocken bereits nach wenigen Zugriffen und auch für längere Dauer zu erfolgen hat.

Eine Maßnahme könnte so lauten: Werden innerhalb einer Stunde für eine Benutzerkennung mehr als 3 Fehlversuche registriert, so wird der Benutzer für 24 Stunden oder bis zum Freischalten durch den Helpdesk gesperrt.

Da eine solche Maßnahme jedoch eine schwerwiegende DoS-Schwachstelle darstellen würde, ist das Problem in diesem Beispiel dadurch zu lösen, dass entweder Passwörter mit einer deutlich höheren Randomness, d.h. größere Länge und Zulassung (fast) beliebiger Zeichen, gefordert werden oder ein Captcha (siehe **M270.1**) abge-

fragt wird.

2.20.2 Blocken von Zugriffen

Das Blockieren von Zugriffen muss nicht zwangsläufig bedeuten, dass der Zugriff (für einen bestimmten Zeitraum) vollständig verboten wird. Wegen der daraus häufig resultierenden DoS-Schwachstelle wird dies sogar eher die Ausnahme darstellen. Stattdessen kommen folgende Maßnahmen in Betracht:

- Verzögerung des Zugriffs
- Verhinderung von automatisierten Zugriffen

Die erste Maßnahme wird im Folgenden beschrieben. Die Maßnahmen zur Verhinderung von automatisierten Zugriffen sind dieselben wie diejenigen zur Verhinderung von DoS-Angriffen und sind in **M270** beschrieben.

Maßname: Verzögerung des Zugriffs

Ein Enumeration-Angriff kann auch dadurch verhindert werden, dass die für die Ausführung eines einzelnen Zugriffs benötigte Zeit so weit in die Länge gezogen wird, dass der Angreifer innerhalb eines vernünftigen Zeitraums nicht die benötigte Anzahl an Zugriffen durchführen kann. Art und Dauer der Verzögerung sind so zu gestalten, dass sie vom legitimen Benutzer nicht als übermäßig störend empfunden werden.

Beispiel 3 – Passwort-Enumeration

Statt wie in Beispiel 2 bereits nach dem dritten fehlerhaften Versuch weitere Zugriffe zu unterbinden, könnte zunächst eine Verzögerung weiterer Versuche erfolgen. Nach dem Absenden erhält der Benutzer die Information "Bitte warten Sie, während ihr Passwort überprüft wird" und erst nach Ablauf einer Minute die positive oder negative Antwort. Ein Blockieren erfolgt dann erst nach 10 Fehlversuchen.

Nachteil einer einfachen Implementierung der *Verzögerungsstrategie* ist die Tatsache, dass der realisierende Prozess oder Thread für die gesamte Dauer des Verzögerungsintervalls am Leben bleibt. Dies macht die Anwendung nun wiederum auf der Implementierungsebene für einen DoS-Angriff anfällig. Durch entsprechend häufiges Auslösen von Verzögerungsthreads könnten die Systemressourcen bis zum Maximum belastet werden. Abhilfe schafft eine entsprechend aufwändige Implementierung in Form eines eigenen Services, der in einem Thread alle wartenden Intervalle überwacht.

Die Verzögerungsstrategie eignet sich in der Regel nicht für Dialoge, bei denen kein eindeutiges Kriterium für den Bezug des Zugriffs auf ein einzelnes Objekt vorliegt (der oben genannte Fall (a)). In obigem Beispiel existiert dieses Kriterium, es ist die Benutzerkennung. Nur der betreffende Benutzer ist von der Verzögerung betroffen. Zum Schutz vor Enumeration von Benutzerkennungen wie in Beispiel 1 ist die Verzögerungsstrategie hingegen nicht oder nur schlecht geeignet, da sie globale Auswirkung hätte und jeder Benutzer, der sich registrieren möchte, der Verzögerung ausgesetzt würde.

Hinweis

Maßnahmen gegen Enumeration-Angriffe führen leicht zur Herstellung von Überflu-

tungsschwachstellen. Beides ist in engem Zusammenhang zu behandeln. Siehe dazu **M270**.

2.21 M290 Sichere Passwörter erzwingen

Abstract **Das System akzeptiert nicht jedes vom Anwender gewünschte Passwort, sondern gibt Regeln vor und prüft die Einhaltung.**

Ebene **Logik**

Beschreibung

Bisher existieren für das Web keine allgemein anerkannten Verfahren und auch keine Standardbibliotheken zur Passwortprüfung. Auch wird häufig die Berücksichtigung anwendungsspezifischer Randbedingungen erforderlich sein. Wir schlagen ein Beispiel vor, das einen guten Kompromiss aus Sicherheit, Einfachheit für den Benutzer und einfacher Überprüfbarkeit darstellt.

Hinweis für den Benutzer

```
Bitte geben Sie hier ein selbst gewähltes Passwort ein. Das Passwort
muss folgenden Regeln entsprechen, die Sie bitte zu Ihrem eigenen
Schutz beachten:

- Mindestlänge: 8 Zeichen (wird überprüft)
- Davon mindestens 2 Ziffern oder Sonderzeichen. Erlaubt sind diese
Sonderzeichen: _,-,#,(,),@,§,! (wird überprüft)
```

Passwortverfahren

```
Tipp: Setzen Sie das Passwort aus 2 falsch geschriebenen Wörtern
zusammen, die Sie mit (mindestens) 2 Sonderzeichen oder Ziffern
verbinden. Beispiele: haus45hoof, albärt##e1stein, fuhss_und_ball,
(strasssen)ban
```

oder

```
Tipp: Ein sicheres Passwort, das sich leicht merken lässt, erhalten
Sie mit diesem Verfahren: Bilden Sie einen Satz und setzen Sie das
Passwort aus den Anfangsbuchstaben in der jeweiligen Groß- oder
Kleinschreibung zusammen. Beispiel: "Unser Haus ist das mit der
schönen Nummer 12": UhidmdsN12, "Am Sonntag stehe ich nie vor 10 Uhr
auf": ASsinv10Ua, "Morgen ist ein Feiertag – das ist aber schön!":
MieF-dias!
```

Eine besonders sichere Implementierung dieses Verfahrens könnte das übergebene Passwort an den Stellen aufspalten, an denen Sonderzeichen stehen, und die einzelnen Fragmente darauf prüfen, ob sie in einer Wortliste vorkommen, und das Passwort in diesem Fall als ungültig ablehnen.

Zusätzlich ist dieser Hinweis angebracht:

```
Bitte verwenden Sie zu Ihrer eigenen Sicherheit dieses Passwort
nicht an anderer Stelle!
```

2.22 M300 Umgang mit Benutzerkennungen

Abstract **Mit Benutzerkennungen sollte sorgsam umgegangen werden.**

Ebene **Semantik, Logik**

Beschreibung

Beim gebräuchlichsten Authentisierungsverfahren im Web, der Authentisierung mittels Benutzerkennung und Passwort, sollte die Benutzerkennung als nicht-öffentliche Information behandelt werden. Bei entsprechendem Schutzbedarf gilt:

- Nicht die E-Mail-Adresse verwenden
- Keine Schemata wie `vorname.nachname` verwenden.

Es wird empfohlen, stattdessen nicht ableitbare Zeichenketten zu verwenden oder eine Kombination aus externen Merkmalen mit einer nicht-erratbaren Komponente, z.B. `peter.meyer.43135`, `kbauer.31244`.

Zudem ist darauf zu achten, dass die Anwendung keine Hinweise über das Vorhandensein von Benutzerkennungen gibt. Häufig anzutreffen ist diese Art der Implementierung einer "Passwort vergessen"-Funktion:

Der Benutzer wird aufgefordert, seine UserID einzugeben. Ist diese korrekt, d.h. existiert die UserID im System, so lautet die Antwort: "Das Passwort wurde an die hinterlegte E-Mail-Adresse verschickt". Wurde jedoch eine nicht existierende UserID eingegeben, so lautet die Antwort: "Dieser Benutzer existiert nicht, bitte korrigieren Sie Ihre Eingabe". Auf diese Weise lassen sich Benutzerkennungen durch Aufzählen (Enumeration) ermitteln. Richtig wäre folgende Antwort, die keine Information über das Vorhandensein einer UserID gibt: "Das Passwort wurde an die hinterlegte E-Mail-Adresse verschickt, falls die eingegebene UserID korrekt gewesen ist. Sollten Sie nicht in Kürze eine E-Mail von uns erhalten, so ist dies darauf zurückzuführen, dass Sie eine fehlerhafte UserID eingegeben haben".

Siehe auch **M280** (Enumeration verhindern).

2.23 M310 Einsatz von SSL/TLS/HTTPS

Abstract **Daten, bei denen Ausspähen verhindert werden muss, sind durch Einsatz des SSL-Protokolls zu schützen.**

Ebene **Implementierung, Technologie**

Beschreibung

Das SSL-Protokoll verschlüsselt die zwischen Browser und Server ausgetauschten Daten und sorgt dafür, dass vertrauliche Informationen vor dem Ausspähen auf dem Übertragungsweg geschützt sind. Diesen Schutz führen heutige Browser soweit, dass auch bei einem Wechsel von einer per SSL geschützten Seite (HTTPS) zu einer ungeschützten Seite (HTTP) – etwa dann, wenn der User auf der SSL-geschützten Seite auf einen HTTP-Link klickt – sensitive Protokollinformationen nicht weitergegeben werden. Dies gilt für:

- Cookies (welche häufig Träger der sensitiven SessionID sind), die mit jedem Browser-Request mitgeschickt werden, solange dieser an denselben Host gerichtet ist bzw. den Pfadrestriktionen entspricht.

- Die Referrer-Information, welche die URL derjenigen Seite enthält, auf der sich der geklickte Link befindet. Dieser liefert Informationen über die der aktuellen Seite vorangegangene Seite mit all ihren GET-Parametern und der SessionID, falls die Technik des URL-Rewritings benutzt wird.

Mit **M170.3** und **M170.4** wird verhindert, dass ein unter SSL gesetztes Cookie bei einem Non-SSL-Zugriff mitgeschickt wird.

Die Übergabe des Referrers mit einem Non-SSL-Zugriff wird von heutigen Browsern dann nicht durchgeführt, wenn die Ausgangsseite per SSL geliefert worden ist. Eine unerwünschte Weitergabe wird damit wirksam verhindert.

Falls jedoch eine Verlinkung auf eine externe Website mittels HTTPS realisiert ist, so wird der Referrer tatsächlich mitgeschickt. **M170.11** ist damit unabhängig davon, ob es sich um eine HTTP- oder HTTPS-Site handelt, anzuwenden. Da ältere Browser die beschriebene Sperre nicht eingebaut haben, ist **M170.11** grundsätzlich zu beachten.

Das SSL Server-Zertifikat stellt außerdem sicher, dass der Hostname authentisch ist. D.h. es garantiert, dass der im Zertifikat angezeigte Hostname unverfälscht ist. Schließen wir den Fall aus, dass das DNS manipuliert worden ist, so garantiert das Zertifikat auch die Echtheit des Hosts (wenn die Seite keine `FRAMES` benutzt).

Auch wenn die Anwendung selbst nicht als schutzbedürftig durch SSL eingestuft wird, sollten folgende Operationen verschlüsselt ablaufen:

- Registrierung

- Login

- Zugriff auf persönliche Daten

- Passwort-Änderung

- "Passwort vergessen"-Funktion

Es sollte berücksichtigt werden, dass auch dann, wenn die Logindaten ausschließlich verschlüsselt übertragen werden, ein Eindringen in eine ansonsten unverschlüsselt betriebene Anwendung durch Ausspähen der SessionID möglich sein könnte.

Bemerkung

Achtung: SSL garantiert nicht die Echtheit einer angezeigten *Webseite*! Ist ein Angreifer in den Server eingedrungen und hat den Inhalt verändert, so wird das Zertifikat nicht verletzt. Von größerer Tragweite als dieser Fall ist jedoch der des Cross-Site Scriptings: Wird eine Webseite mittels Cross-Site Scripting verändert oder durch eine komplett gefälschte Seite überdeckt, so ist dies ebenfalls nicht mit einer Verletzung des Zertifikats verbunden.

2.24 M320 Information Disclosure verhindern

Abstract **Im Folgenden werden Maßnahmen gegen eine ungewollte Veröffentlichung interner Informationen zusammengefasst.**

Ebene **Implementierung, Logik, Semantik**

Beschreibung

Zur Einschränkung von Information Disclosure können folgende Ansatzpunkte herangezogen werden:

- Aus der HTML-Seite sämtliche Kommentare entfernen.
- Fingerprints entfernen:
  - Open Source Module umbenennen.
  - Server-Banner entfernen.
  - Typische Merkmale wie Meta-Tags, Form-Variablennamen usw. umbenennen.
- Fehlermeldungen abfangen und dafür neutrale Fehlerseiten liefern.
- Open Source- oder sonstige Fremdsoftware daraufhin untersuchen, ob sie im Fehlerfall sicherheitsrelevante Informationen ausgeben.
- Auf richtige Serverkonfiguration achten. Siehe **M370**, **M380**, **M390.**
- Enumeration verhindern. Siehe **M280**.

Bemerkung

In Bezug auf SQL-Injection ist das Abfangen von Fehlermeldungen keinesfalls ausreichend. Beim Vorhandensein einer SQL-Injection-Schwachstelle können mit geschickter Abfragetechnik allein aus der Tatsache, dass die Anwendung in einem Fall eine Ausgabe, im anderen Fall eine anonyme Fehlerseite liefert, bereits weitreichende Schlüsse gezogen werden.

2.25 M335 Monitoring und Patching

Abstract **Eingesetzte Standard- und Systemsoftware ist auf Known Vulnerabilities zu überwachen. Logdateien sind zu monitoren.**

Ebene **System**

Beschreibung

Unternehmen haben heutzutage in der Regel Prozesse installiert, die mögliche Sicherheitsprobleme der im Einsatz befindlichen Systeme überwachen, und bei Entdeckung von Schwachstellen oder Auftreten von Sicherheitsvorfällen geeignet reagieren. Webserver und sonstige von den Webanwendungen verwendete Systeme sind in diese Prozesse aufzunehmen. Dabei sind auch Backendsysteme im internen Netz einzubeziehen, sofern Webanwendungen auf sie zugreifen. Denn bei unzureichender Prüfung der übergebenen Daten durch die Webanwendungen könnten auch diese Systeme über das Internet in unvorhergesehener Weise zugänglich sein.

Zusätzlich empfiehlt sich der regelmäßige Einsatz sog. Web Application Security Scanner, die bei der Erkennung von Known Vulnerabilities sehr gute Dienste leisten.

Das schnelle Einspielen eines Patches ist sicher die wirksamste Schutzmaßnahme. Gerade in komplexen Umgebungen ist dies jedoch wegen möglicher Seiteneffekte nicht immer zeitnah möglich. Ersatzweise sollte der Sicherheitsprozess dann eine Risikoanalyse beinhalten, an deren Ende die Umsetzung angemessener alternativer Maßnahmen steht. Die Anwendungsverantwortlichen sollten von den Systemverantwortlichen, die in der Regel für die hier genannten Prozesse verantwortlich sind, umgehend über die geänderte Sicherheitslage informiert werden.

M335.1 **Logdateien kontrollieren**

Alle Log- und Protokolldateien sollten regelmäßig kontrolliert und ausgewertet werden. Hierbei müssen neben den fachlichen Aspekten auch rechtliche, z.B. des Datenschutzes, beachtet werden. Die Dateien sind insbesondere auf Auffälligkeiten zu prüfen. Die folgende Liste erhebt keinen Anspruch auf Vollständigkeit. Sie soll Anhaltspunkte geben, wie man Hinweise auf Sicherheitslücken, Angriffe oder Manipulationen erkennt:

- Größe der Logdatei, Anzahl der Meldungen
- Häufigkeit von Meldungen zu bestimmten Zeiten
- Häufigkeit von Meldungen von bestimmten Source-IPs
- häufige fehlerhafte Anmeldeversuche
- häufige fehlerhafte URLs
- viele sehr ähnliche URLs (z.B. nur einzelne Zeichen verschieden)
- ungültige Parameter-Namen und/oder Werten

- offensichtlich manipulierte Daten (z.B. von `Hidden`-Parametern)
- URLs mit Path Traversal

Logdateien sollten nicht mit einem Browser betrachtet werden. Ebenso sollten die Logdateien der Server selbst, und nicht davon abgeleitete und speziell aufbereitete Dateien inspiziert werden.

2.26 M340 Systemkonfiguration: Benutzerkennung für Web-/Applicationserver unter UNIX

Abstract **Web- und Applicationserver sollen jeweils mit einer eigenen Benutzerkennung laufen. Diese und die zugehörige Gruppenkennung ist im Betriebssystem entsprechend zu konfigurieren.**

Ebene **System**

Beschreibung

Wenn nicht besondere Gründe dagegen sprechen, sollte jeder Web- bzw. Applicationserver mit einer eigenen Benutzer- und Gruppenkennung laufen. Bei UNIX-Standardinstallationen wird dafür oft `nobody` verwendet. Davon ist abzuraten, da `nobody` auch für viele andere Dienste verwendet wird. Zum einen würden diese Dienste bei einem erfolgreichen Angriff auf den Webserver unnötig kompromittiert werden, und zum anderen könnten diese Dienste dann selbst als Ausgangspunkt für einen Angriff auf den Webserver dienen.

Werden mehrere Instanzen eines Web- oder Applicationservers auf einem Rechner betrieben, dann sollte jede Instanz unter einer eigenen Benutzerkennung laufen. Unter UNIX ist das einfach mit dem Standard-Benutzer-/Gruppen-Konzept realisierbar.

Die hier definierten Benutzerkennungen müssen dann bei der Konfiguration der Server verwendet werden, siehe dazu **M370**, etc.

Beispiele

`/etc/passwd` unter UNIX/Linux:

```
wwwrun:x:4711:4711:root Web Server:/ServerRoot:/sbin/nologin
wwwfoo:x:4712:4711: foo Web Server:/DocRoot/foo:/sbin/nologin
wwwbar:x:4713:4711: bar Web Server:/DocRoot/bar:/sbin/nologin
```

`/etc/group` unter UNIX/Linux:

```
wwwrun:x:4711:wwwrun,wwwfoo,wwwbar
```

Eine detaillierte Beschreibung findet sich hierzu auch in der Apache Webserver Sicherheitsstudie des BSI (siehe [13]).

Bemerkungen

Wenn der Server Anfragen auf privilegierten Ports (1 bis 1023) bearbeiten soll, dann muss der Serverprozess unter UNIX als Benutzer `root` gestartet werden. In der Konfiguration der Server (siehe z.B. **M370**) wird dann festgelegt, als welcher Benutzer der Prozess wirklich läuft.

Das weit verbreitete Apache-Modul `mod_perl` wird z.B. mit der `PerlRequire` Directive in `httpd.conf` initialisiert. Diese Initialisierung wird als Benutzer `root` durchgeführt, damit könnten dann auch alle weiteren Berechtigungen, z.B. in diesem Modul initialisierte Datenbankverbindungen, an `root` gebunden sein. Kann der Server mit einem nicht-privilegierten Port arbeiten, dann kann der Prozess sofort unter der gewünschten Benutzerkennung gestartet werden. Für sehr sicherheitskritische Anwen-

dungen sollte man also erwägen, einen entsprechenden Port zu verwenden.

2.27 M350 Systemkonfiguration: Dateiberechtigungen

Abstract **Verzeichnisse und Dateien müssen mit Zugriffsrechten ausgestattet werden, die nur dem autorisierten Benutzer die erforderlichen Berechtigungen geben**

Ebene **System**

Beschreibung

Die Dateien und Verzeichnisse des Web- bzw. Applicationserver werden mit entsprechenden Berechtigungen versehen. Dabei ist zwischen den Daten und Programmen für den Dienst selbst (z.B. Programme, Skripte, Konfiguration) und den Anwendungsdaten (z.B. `/DocumentRoot`) zu unterscheiden.

Die Berechtigungen werden unter Voraussetzung von **M340** gesetzt.

Eine detaillierte Beschreibung findet sich hierzu auch in den Apache- und IIS-Webserver-Sicherheitsstudien des BSI (siehe [13][14]).

Beispiele

Datei- und Verzeichnisberechtigungen[4] für UNIX/Linux:

```
chown -R wwwrun:wwwrun /ServerRoot
chown    root:root     /ServerRoot
chown    root:root     /ServerRoot/log
find /ServerRoot   -type d -exec chmod 2750 {} \;
find /ServerRoot   -type f -exec chmod o-x,o-r,o-w {} \;
find /ProgramRoot  -type d -exec chmod 2550 {} \;
find /ProgramRoot  -type f -exec chmod 550 {} \;
find /DocumentRoot -type d -exec chmod 2550 {} \;
find /DocumentRoot -type f -exec chmod 440 {} \;
```

Entgegen der bei Apache üblichen Ablage der Logdateien in `/var/log/httpd` werden im obigen Beispiel die Logdateien in `/ServerRoot/logs` erwartet. Dort sollte nur `root` Lesen und Schreiben können.

Achtung: die Reihenfolge obiger Kommandos ist wichtig.

Datei- und Verzeichnisberechtigungen für Windows:

```
cd ServerRoot
cacls.exe . /T /E /R TERMINALSERVERBENUTZER
cacls.exe . /T /E /R Benutzer
cacls.exe . /T /E /G wwwrun:F
cacls.exe . /T /E /G wwwgrp:R
```

Der `TERMINALSERVERBENUTZER` ist hier explizit aufgeführt, weil dieser Benutzer von Windows automatisch angelegt und in verschiedenen Gruppen eingetragen wird, wenn die Terminaldienste installiert sind.

[4] Das Unix/Linux Programm chmod wird hier einmal mit numerischen Rechten (550) und einmal mit symbolischen Rechten (o-x) verwendet. I.d.R. ist die Verwendung der numerischen Rechte sicherer. Manchmal müssen aber besondere Berechtigungen, die z.B. bei der Installation vergeben wurden, beibehalten werden. Dann muss man mit den symbolischen Rechten arbeiten.

Bemerkungen

Unter UNIX/Linux lassen sich die Berechtigungen noch restriktiver setzen. Durch die Maßnahme **M370** und **M340** ist sichergestellt, dass der Webserver unter seiner eigenen Benutzerkennung läuft, dann sind nur noch Berechtigungen für den Besitzer der Verzeichnisse und Dateien, nicht aber für die Gruppe nötig. Statt `2750` und `550` kann dann `2700` und `500` bei `chmod` verwendet werden.

Unter Windows ist es schwieriger, ein allgemein gültiges Vorgehen zu definieren, da die existierenden Berechtigungen stark von den Systemeinstellungen in Windows selbst abhängen. Zusätzlich muss man immer mit `calcs.exe` überprüfen, welche zusätzlichen Berechtigungen vergeben sind und diese dann ebenfalls entsprechend ändern.

M350.1 Spezielle Dateiberechtigungen

Für besondere Anwendungsfälle müssen zum oben Gesagten abweichende Zugriffsrechte für Verzeichnisse und Dateien definiert werden.

`Besonders strikte Berechtigungen`

Alle statischen Inhalte (also Dateien, die nur lesend gebraucht werden) sollten mit sehr strikten Zugriffsberechtigungen ausgestattet werden. Statisch in diesem Sinne sind auch immer alle Programm- und Konfigurationsdateien.

In solchen Umgebungen kann es sinnvoll sein, dass alle Verzeichnisse und Dateien dem Benutzer `root` gehören. Dann müssen sie von der Gruppe lesbar sein, wobei der Benutzer des Webservers Mitglied dieser Gruppe sein muss.

`Eigene Benutzerkennung pro Instanz`

Programm-Datei- und Verzeichnisberechtigungen für UNIX/Linux:

```
find /ProgramRoot  -type d -exec chmod 2500 {} \;
find /ProgramRoot  -type f -exec chmod 500 {} \;
find /DocumentRoot -type d -exec chmod 2500 {} \;
find /DocumentRoot -type f -exec chmod 400 {} \;
```

`Berechtigungen bei mehreren Instanzen`

Bei mehreren Instanzen derselben Software (also z.B. eine Installation der Webserver Software, von der aber mehrere Prozesse gestartet werden) ist es besonders wichtig, die Berechtigungen der Datenzugriffe gegeneinander abzusichern, da die Software für jede Instanz die gleichen Berechtigungen hätte. Die Dienste laufen unter der Benutzerkennung `wwwfoo` bzw. `wwwbar`, darum muss die Software (`/ServerRoot/`) für die Gruppe lese- und ausführbar sein. Die Daten dürfen aber nur für den Besitzer lesbar sein. Die oben unter **M350** genannten Empfehlungen sind daher wie in den folgenden Beispielen angegeben, abzuändern.

Beispiele

Programm-Datei- und Verzeichnisberechtigungen unter UNIX:

(1) Zunächst muss die Software für die Gruppe lese- und ausführbar sein:

```
# (1) ServerRoot für die Gruppe
chown -R wwwrun:wwwrun /ServerRoot
find /ServerRoot  -type d -exec chmod 2750 {} \;
find /ServerRoot  -type f -exec chmod 550 {} \;
```

(2) Dann wird für jede Instanz das Verwaltungsverzeichnis (für Logdateien, etc.) mit Berechtigungen ausschließlich für den Benutzer der Instanz versehen

```
# (2) Verwaltungsverzeichnis pro Instanz
chown wwwfoo:wwwrun /ServerRoot/wwwfoo
chown wwwbar:wwwrun /ServerRoot/wwwbar
find /ServerRoot/wwwfoo -type d -exec chmod 2700 {} \;
find /ServerRoot/wwwbar -type d -exec chmod 2700 {} \;
```

(3), (4) `/ProgramRoot` für die Programme und Skripte der Instanz ist ebenso wie `/DocumentRoot` mit Berechtigungen ausschließlich für den Benutzer der Instanz zu versehen.

```
# (3) ProgramRoot pro Instanz
chown wwwrun:wwwrun /ProgramRoot
chmod 2550 /ProgramRoot
chown -R wwwfoo:wwwrun /ProgramRoot/foo
find /ProgramRoot/foo -type d -exec chmod 2500 {} \;
find /ProgramRoot/foo -type f -exec chmod 400 {} \;
chown -R wwwbar:wwwrun /ProgramRoot/bar
find /ProgramRoot/bar -type d -exec chmod 2550 {} \;
find /ProgramRoot/bar -type f -exec chmod 400 {} \;

# (4) DocumentRoot pro Instanz
chown wwwrun:wwwrun /DocRoot
chmod 2550 /DocRoot
chown -R wwwfoo:wwwrun /DocRoot/foo
find /DocRoot/foo -type d -exec chmod 2500 {} \;
find /DocRoot/foo -type f -exec chmod 400 {} \;
chown -R wwwbar:wwwrun /DocRoot/bar
find /DocRoot/bar -type d -exec chmod 2550 {} \;
find /DocRoot/bar -type f -exec chmod 400 {} \;
```

Bemerkungen

Bei Vorliegen von mehreren Instanzen muss man sich entscheiden, ob die zugehörigen Konfigurationsdateien im gruppenzugänglichen `/ServerRoot` liegen oder in dem Verwaltungsverzeichnis der Instanz. In beiden Fällen sollten die Konfigurationsdateien nur lesbar für den Benutzer sein (`chmod 400` in UNIX).

2.28 M360 Serverkonfiguration: Environment und Startup

Abstract **Serverprozesse sollen mit einem genau definierten Environment ausgestattet werden. Startup-Skripte müssen dieses gewährleisten und auch genau den von ihnen selbst gestarteten Server beenden können.**

Ebene **System**

Beschreibung

Der Startkontext eines Servers ist sicherheitsrelevant. So erbt ein Prozess von seinem aufrufenden Parent-Prozess (in UNIX meist eine Shell) das gesamte Environment. Es gilt also, dieses Environment zu bereinigen bzw. vollständig neu zu initialisieren.

Da die meisten Dienste mit einem rc-Skript gestartet werden, ist genau dies die richtige Stelle für die Anpassungen.

Für Start und Stopp eines Server-Dienstes muss ein entsprechendes Startup-Skript erstellt werden. Dabei sind die Vorgaben bzgl. Environmentvariablen zu beachten. Weiter ist wichtig, dass der Dienst ordnungsgemäß beendet wird. Einige Produkte bringen dazu ihre eigenen Skripte mit, diese sind dann zu überprüfen. Andere Produkte haben keine Skripte, für sie könnte das folgende Beispiel verwendet werden.

Beispiele

rc-Skript für Apache mit PHP auf UNIX/Linux.

```
#!/bin/sh

# (1) Environmentvariablen definieren
PATH=/bin:/usr/bin:/ServerRoot/bin:/opt/php/bin:
FS="   "                              # space, h-tab, newline
SHELL=/bin/sh

# (2) alle anderen löschen
_RO='BASH_VERSINFO|EUID|PPID|UID|SHELLOPTS'
_OK='IFS|SHELL|PATH|LANG|TZ'
unset `\set|/bin/awk -F= '/^('$_RO'|'$_OK')=/{next}{printf " "$1}'`
unset _RO _OK
USER=nsuser
HOME=/tmp
PHPRC=/ServerRoot/conf

# (3) start/stop des Dienstes
\cd /ServerRoot/bin/
case "$1" in
 'start')
    /bin/echo -n "starting http ... "
    ./httpd -f /ServerRoot/conf/httpd.conf && \
    /bin/echo httpd started
 'stop')
    /bin/echo -n "stopping httpd ... "
    pid=`/bin/ps -ef|/bin/awk '($8=="./httpd"&&$3==1){printf " $2}'`
    if [ -n "$pid" ]; then
         /bin/kill $pid && /bin/echo stopped
    else
         /bin/echo httpd not running
    fi
    ;;
```

```
esac
exit $?
```

Erklärung des Beispiels:

(1) Reduzierung des Environment auf das Notwendigste: PATH, SHELL, USER, HOME, FS, PHPRC

(2) Einige Shells haben read-only Variablen; um Fehlermeldungen in diesem Skript zu vermeiden, dürfen sie nicht gelöscht werden. Diese Liste der Variablen ist je nach Shell anzupassen.

(3) Zum Starten des Servers wird kein vollständiger Pfad verwendet, dieser ist nur diesem Skript bekannt, und kann somit nicht einfach mit ps festgestellt werden

2.29 M370 Serverkonfiguration: Webserver

Abstract **Ein Webserver muss in Bezug auf Benutzerkennung, Zugriffsrechte auf Prozesse, Verzeichnisse und Dateien, Fehlerausgaben und Protokollierung sicher konfiguriert werden.**

Ebene **System**

Beschreibung

Nach der Installation des Servers müssen Sicherheitseigenschaften angepasst werden. Die weiteren Maßnahmen gehen davon aus, dass die Software vollständig installiert ist. Es wird weiter vorausgesetzt, dass alle zum Betrieb der Applikation nicht relevanten Teile, wie z. B. Samples, Manuals und Testseiten, bereits entfernt sind.

Eine sichere Konfiguration umfasst dann mindestens folgende 4 Konzepte:

`Eingeschränkte Benutzerkennung und Zugriffsrechte`

Die Benutzer- und Gruppenkennung muss bereits im Betriebssystem vorgesehen sein, siehe dazu **M340**, **M350**. Diese niedrig-privilegierten Kennungen werden dann benutzt, um den Serverprozess auszuführen. Dies reduziert die Gefahr, dass durch einen kompromittierten Server weitere Teile des Systems gelesen oder verändert werden können.

`Keine unnötige Veröffentlichung interner Informationen, Fehlerausgabe`

Wenn ein Server sich mit einem ausführlichen Server-Banner meldet, dann lässt sich daraus unmittelbar auf ggf. vorhandene Schwachstellen schließen. Fehlt dagegen eine ausführliche Information über den Typ des Servers, dann müssten entsprechende Tests durchgeführt werden. Ein Abschalten des Server-Banners und anderer Informationen erhöht die Sicherheit. Nicht zu vergessen sind die Fehlerseiten des Webservers, die bei bestimmten Fehlern automatisch vom Webserver ausgeliefert werden; auch sie sind so zu gestalten, dass sie keine weiteren Informationen über das System und den Webserver bekannt geben (Fehlerausgaben der Applikation werden in **M250** behandelt).

`Getrennte Verzeichnisse für Programme und Daten`

Die Verzeichnisse, die ein Webserver für Programme, Skripte und Daten (statische Seiten) verwendet, sind zu trennen, so dass z.B. Zugriffe auf Seiten keine Skripte im Sourcecode ausliefern.

`Gute Protokollierung`

Die Protokollierung ist so zu konfigurieren, dass jederzeit alle Aktionen (Zugriffe) und Transaktionen einsehbar sind. Ein Administrator muss in der Lage sein, Angriffe, Angriffsversuche, und Verwendung von problematischem Code durch Auswertung der Logdateien zu erkennen. Siehe auch **M335.1**.

Beispiele

Die folgenden Beispiele betrachten die `http.conf` für Apache. Weitere detaillierte

Beschreibungen und Beispiele finden sich in der Apache Webserver Sicherheitsstudie des BSI (siehe [13]).

(1) Jede Webserver-Instanz hat eine eigene Benutzer- und Gruppenkennung, mit der der Serverdienst gestartet wird. Die Gruppenkennung kann mit der anderer Dienste (z.B. Applicationserver) geteilt werden, wenn beide auf die gleichen Daten und/oder Verzeichnisse zugreifen müssen.

```
# (1) Benutzer-, Gruppenkennung
User            wwwrun
Group           wwwgrp
```

(2) Die Server-Signatur und anderer Informationen im Apache werden abgeschaltet, um die Sicherheit zu erhöhen.

```
# (2) Server Signatur
ServerSignature Off
ServerTokens    ProductOnly
ExtendedStatus  Off
```

(3) Die Verzeichnisse für die Server-Software und die Dokumente sowie Programme/Skripte des Webservers sind strikt getrennt. Idealerweise sind mindestens für `DocumentRoot` und `ServerRoot` jeweils eigene Partitionen vorgesehen.

```
# (3) Verzeichnisse
ServerRoot      /ServerRoot
DocumentRoot    /DocumentRoot/htdocs
ScriptAlias     /cgi-bin/ "/ProgramRoot/"
<Directory "/ProgramRoot">
  Options       None
  AllowOverride None
  Order         allow,deny
  Allow from    all
  <LIMIT GET POST>
     Order      allow,deny
     Allow from all
  </LIMIT>
  <LIMITExcept GET POST>
     Order      deny,allow
     Deny from  all
  </LIMIT>
</Directory>
```

(4) Die Zugriffsrechte für DocumentRoot werden zunächst sehr restriktiv gesetzt. Mit `Deny from all` ist sichergestellt, dass nur noch Zugriffe erlaubt sind, die mit einem entsprechenden `Allow from ..` freigegeben wurden. Die `LIMIT*`-Werte sind entsprechend den Anforderungen anzupassen. Danach werden die Zugriffsrechte für den physikalischen Pfad zu `DocumentRoot` explizit gesetzt. Insbesondere werden in `DocumentRoot` keine ausführbaren Dateien (z. B. CGI) erlaubt, und in Server-Side-Includes (SSI) ist es nicht möglich, externe Programme aufzurufen. Außerdem sind die Zugriffsmethoden auf `HEAD`, `GET` und `POST` beschränkt.

```
# (4) Berechtigungen für Verzeichnisse
<Directory />
  Options       None
  AllowOverride None
  LimitRequestBody   204800
  LimitRequestFields 50
  Order         deny,allow
  Deny from     all
</Directory>
```

```
<Directory "/DocumentRoot/htdocs">
  Options       +FollowSymLinks -ExecCGI -IncludesNOEXEC
  AllowOverride None
  <LIMIT HEAD GET POST>
     Order      allow,deny
     Allow from all
  </LIMIT>
  <LIMITExcept HEAD GET POST>
     Order      deny,allow
     Deny from  all
  </LIMIT>
</Directory>
```

(5) Die Konfigurationsdateien des Webservers müssen gegen unberechtigten Zugriff, vor allem über URLs, geschützt werden. Dazu wird der Name der Dateien festgelegt, und anschließend der Zugriff verweigert. Weiter wird der Zugriff auf häufig *vergessene* Dateien verweigert, diese werden an ihren Dateinamenerweiterungen erkannt.

```
# (5) Zugriffsschutz auf bestimmte Dateien
AccessFileName  .htaccess
<Files ~ "^\.ht">
  Order         allow,deny
  Deny from     all
  Satisfy       all
</Files>
<FilesMatch "\.(old|bak|tar|tgz|gz|inc|cfg|conf)$">
  Order         allow,deny
  Deny from     all
</Files>
```

(6) Zugriff auf Benutzerverzeichnisse wird deaktiviert.

```
# (6) Zugriffsschutz auf Benutzerverzeichnisse
UserDir         disabled
# oder
UserDir         /DocumentRoot/htdocs
```

(7) Die Größe der Daten, die ein Benutzer zum Server schicken kann, werden auf einen vernünftigen Wert begrenzt, um DoS-Angriffe zu verhindern. Welcher Wert *vernünftig* ist, entscheidet die Applikation, also z.B. "maximale Größe von Dateien, die gesendet werden" oder "maximale Größe der Parameter aus Formularen".

```
# (7) Größe von Uploads begrenzen
SetEnvIf Content-Length "[1-9][0-9]{4,}" too_large=1
<Loction /upload>
  Order         Deny,Allow
  Deny from     env=too_large
  ErrorDocument 403 /DocumentRoot/htdocs/upload_too_large.html
</Location>
```

(8) Auch die Logdateien liegen in einem Verzeichnis, welches nicht über `DocumentRoot` erreicht werden kann. Empfohlene Zugriffsrechte für dieses Verzeichnis und die Dateien siehe **M340**, **M350**, **M350.1**.

```
# (8) Log- und sonstige Dateien
LockFile        /ServerRoot/var/lock
PidFile         /ServerRoot/var/pid
ScoreBoardFile  /ServerRoot/var/score
ErrorLog        /ServerRoot/log/error_log
CustomLog       /ServerRoot/log/access_log
```

Bemerkung

zu (6): mit `disabled` alleine ist zwar der Zugriff nicht mehr möglich, aber der Webserver gibt durch unterschiedliche Fehlerseiten noch bekannt, ob der User existiert oder nicht. Will man vermeiden, dass Enumeration der Benutzernamen erfolgt, dann ist die zweite Variante zu wählen.

zu (7): man beachte, dass die Verwendung der Environmentvariablen `CONTENT_LENGTH` in der Applikation nicht verhindert, dass große Datenmengen gesendet werden, da der Webserver die Daten zuerst vollständig annimmt, bevor `CONTENT_LENGTH` gesetzt wird, und dann die Applikation aufgerufen wird.

M370.1 Namenserweiterungen

Ein Webserver erkennt an der Dateinamenserweiterung (File-Extension), um welche Art von Datei es sich handelt, und wie diese (an den Browser) ausgeliefert wird. Fehlt eine Zuordnung, dann darf der Webserver die angeforderte Datei nicht ausliefern. Häufig verwenden Zusatz-Programme wie z.B. Perl oder PHP bestimmte Includedateien. Um zu verhindern, dass diese unbeabsichtigt gelesen werden, ist deren Endung so zu wählen wie die des Programms/Skripts, welches die Datei inkludiert. Dadurch wird automatisch verhindert, dass solche Dateien irrtümlich (z.B. nach Änderung der Serverkonfiguration) als Text ausgeliefert werden.

Beispiele

```
# (1) allgemein
global.inc.cgi

# (2) Perl
global.inc.pl

# (3) PHP
global.inc.php

# (4) ASP
global.inc.asp
```

Bemerkung

Diese Konfiguration verhindert nicht, dass die Datei direkt via URL aufgerufen wird. Ein solcher direkter Aufruf einer Includedatei kann zu einem nicht vorhersehbaren Verhalten führen, da die Funktion der meisten Includedateien an bestimmte Voraussetzungen geknüpft ist, die beim direkten Aufruf unter Umständen nicht gegeben sind. Um den Direktaufruf vollständig zu verhindern, müssen die Includedateien in ein eigenes Verzeichnis außerhalb von DocumentRoot bzw. ProgramRoot gelegt werden. Die Anwendungen sind entsprechend anzupassen.

M370.2 Fehlerseiten

Ein Webserver liefert beim Erkennen von Fehlersituationen automatisch entsprechende Seiten, meist passend zum Fehler, aus. Diese Seiten können (meist) frei gestaltet werden. Dabei ist darauf zu achten, dass keine unnötigen Informationen in der Fehlerseite enthalten sind, insbesondere keine Pfadnamen. Außerdem ist sicherzustellen, dass keine Eingabedaten (URL oder HTTP-Header) ausgegeben werden. Wenn die Ausgabe von solchen Parametern doch notwendig ist, dann müssen diese entspre-

chend gefiltert werden, siehe entsprechende *Data Validation* Maßnahmen **M100** bis **M150**

.

2.30 M380 Serverkonfiguration: Datenbankserver

Abstract **Ein Datenbankserver muss in Bezug auf Benutzerkennung, Zugriffsrechte auf Datenbanken, Tabellen, Stored Procedures und Tabelleninhalte sicher konfiguriert werden.**

Ebene **System**

Beschreibung

In den Maßnahmen **M340** und **M350** wurde bereits erklärt, wie die Prozesse, Verzeichnisse und Dateien mit entsprechenden Benutzerkennungen zu versehen sind, so dass der mögliche Schaden bei einem Eindringen in die Webanwendung durch eine zweite Verteidigungslinie begrenzt wird. Dasselbe Prinzip ist durch entsprechende Konfiguration auch bei Datenbanken anzuwenden. Diese Konfiguration umfasst mindestens folgende 4 Konzepte:

`Eigener Administrationsbenutzer`

Traditionell verwalten Datenbanken ihre eigenen Benutzer. Diese sind unabhängig von den Benutzern des Host-Systems. Für die Benutzerverwaltung verwenden die meisten Datenbanksysteme eine eigene Tabelle oder sogar eine eigene Datenbank. Zugriff auf diese Tabelle hat normalerweise nur der voreingestellte Benutzer `root` (oder `administrator` oder `sa`). Dieser Benutzer hat automatisch auch auf alle anderen Datenbanken Zugriff. Er darf nicht von der Applikation für Zugriffe auf ihre Daten verwendet werden.

`Sicherer Zugang und Passwort für Administrationsbenutzer`

Für den Administrationsbenutzer ist ein Passwort zu setzen. Außerdem sollte dieser nur vom localhost aus zugreifen dürfen.

`Systemtabellen und Systemfunktionen sichern`

Zugriff auf die Systemtabellen und Systemfunktionen (z.B. *Stored Procedures*) müssen soweit wie möglich verboten werden.

`Applikationsspezifische Benutzer mit jeweils eigenem Passwort`

Wenn auf eine Datenbank-Instanz mehrere Applikation zugreifen, dann ist für jede Applikation eine eigene Benutzerkennung zu verwenden. Diese einzelnen Benutzer sollten nur auf ihre eigenen Daten zugreifen können (sofern die Applikation nichts anderes erfordert).

Beispiele

`MySQL`

Die folgenden Beispiele betrachten eine MySQL Datenbank:

(1) Eigener Administrationsbenutzer

Im Allgemeinen ist eine Administratorkennung bereits vorhanden, sie sollte zur Sicherheit aber überprüft werden:

```
USE mysql;
SELECT * FROM user WHERE User='root';
```

(2) Sicherer Zugang und Passwort für Administrationsbenutzer

```
USE mysql;
UPDATE user SET Host='localhost'            WHERE User='root';
UPDATE user SET Password=password('geheim') WHERE User='root';
```

(3) Systemtabellen und Systemfunktionen sichern

Bei MySQL müssen hier nur die Rechte in der `mysql.user` Tabelle gesetzt werden:

```
USE mysql;
UPDATE user SET Select_priv='N'       WHERE NOT (User='root');
UPDATE user SET Insert_priv='N'       WHERE NOT (User='root');
UPDATE user SET Update_priv='N'       WHERE NOT (User='root');
UPDATE user SET Delete_priv='N'       WHERE NOT (User='root');
UPDATE user SET Create_priv='N'       WHERE NOT (User='root');
UPDATE user SET Drop_priv='N'         WHERE NOT (User='root');
UPDATE user SET Reload_priv='N'       WHERE NOT (User='root');
UPDATE user SET Shutdown_priv='N'     WHERE NOT (User='root');
UPDATE user SET Process_priv='N'      WHERE NOT (User='root');
UPDATE user SET File_priv='N'         WHERE NOT (User='root');
UPDATE user SET Grant_priv='N'        WHERE NOT (User='root');
UPDATE user SET Index_priv='N'        WHERE NOT (User='root');
UPDATE user SET Alter_priv='N'        WHERE NOT (User='root');
```

(4) Applikationsspezifische Benutzer mit jeweils eigenem Passwort

Zunächst werden alle schwach gesicherten Benutzer gelöscht:

```
USE mysql;
DELETE FROM user WHERE Host='' OR Host='%';
DELETE FROM user WHERE User='' OR User='%' OR Password='';
DELETE FROM db   WHERE Host='' OR Host='%';
```

```
USE mysql;
INSERT INTO user SET
Host='www.domain.tld',User='wwwsql',Password=password('sicher');
INSERT INTO db   SET Host='www.domain.tld',User='wwwsql',Db='www';
```

Microsoft SQL Server

(3) Systemtabellen und Systemfunktionen sichern

Beispielhaft sei erklärt, wie in der Registry die Verwendung der `xp_*` (z. B. `xp_cmdshell`) Stored Procedures abgeschaltet werden:

```
HKLM\Software\Microsoft\Microsoft SQL Server\<Instanz
Name>\Providers\DisallowAdhocAccess
```

oder:

```
HKLM\Software\Microsoft\MSSQLServer\[ccc] MSSQL\DisAllowAdhocAccess
```

Für eine sichere Konfiguration des Microsoft SQL Server sei auf die ausführliche Dokumentation von Microsoft selbst oder auf [15] verwiesen.

Bemerkungen

Eine häufig anzutreffende Empfehlung zur sicheren Konfiguration des Datenbankservers ist, den Server auf demselben Rechner wie den Web- und/oder Applicationserver zu betreiben. So könnten dann Remotezugriffe auf den Datenbankserver vermieden werden; die Webanwendung könnte via localhost zugreifen. Diese Empfehlung bringt jedoch Gefahren mit sich. Zum Beispiel wird vergessen, dass der Datenbankserver seine Datenbanken und Tabellen als Dateien ablegt. Diese Dateien sind dann wiederum Angriffen auf das Dateisystem ausgesetzt, was die Konfiguration des Datenbankservers nicht effektiv verhindern kann. Mit einer XSS-, Command-Injection- oder SQL-Injection-Schwachstelle könnten die Datenbankdateien selbst gestohlen oder manipuliert werden. Wenn immer möglich sollten die verschiedenen Dienste (hier Webserver und Datenbankserver) auf getrennten Rechnern laufen, siehe dazu **M400** (Web Application Isolation).

2.31 M390 Serverkonfiguration: PHP-Umgebung

Abstract **PHP bietet eine Vielzahl von Konfigurationsmöglichkeiten, um bekannten Schwachstellen zu begegnen oder sie auszuschließen.**

Ebene **System**

Beschreibung

Die Konfiguration einer sicheren PHP-Umgebung ist auch für erfahrene Administratoren eine Herausforderung. PHP ist zwar nicht prinzipiell unsicher, jedoch ist die Grundkonfiguration eher auf maximale Benutzbarkeit ausgelegt.

Leider fehlt PHP ein globaler Schalter, ähnlich wie *tainted mode* in Perl, so dass man viele Einstellungen entweder selbst in der Konfigurationsdatei `php.ini` oder im Applikationscode vornehmen muss. `php.ini` enthält viele sicherheitsrelevante Konfigurationsoptionen, die auch meist gut dokumentiert sind.

Zu den problematischen Optionen gehört `register_globals=On`. Obwohl seit Version 4.2.0 standardmäßig `Off`, wird sie oft wieder auf `On` gesetzt, um ältere Anwendungen nicht anpassen zu müssen. Während die Einstellung `On` häufig zu Schwachstellen führen kann, fördert `register_globals=Off` die genauere Prüfung von Eingabedaten durch den Programmierer (s.a. **M150 Data Validation**).

Im Weiteren problematisch sind die Anweisungen `include` und `require`. Bei unsachgemäßer Nutzung können sie zum Nachladen von Dateien aus beliebiger Quelle missbraucht werden.

PHP bietet außerdem eine einfache API zur Verwaltung von Sessions. Jedoch gibt es in der Standardkonfiguration eine Reihe von Problemen, insbesondere in Bezug auf die Nutzung von SessionIDs.

Im Folgenden werden die wichtigsten Einstellungen für die `php.ini` erklärt.

Beispiele

(1) Server Signatur und Logdateien

Abschalten der Server-Signatur und anderer Informationen, aber Protokollierung aller Fehler in der Logdatei:

```
[PHP]
  expose_php          = Off
  display_errors      = Off
  error_reporting     = E_ALL
```

(2) Variablen sichern

PHP sucht in verschiedenen Quellen nach der Belegung von Variablen. Die Suchreihenfolge ist standardmäßig festgelegt. Die Suche ist beendet, sobald die gewünschte Variable gefunden ist. Ein Programmierer muss selbst sicherstellen, dass die Variable aus der richtigen Quelle gelesen wird. PHP kann die Liste der für die Anwendung erlaubten Quellen vorgeben. Dadurch lassen sich Manipulationsversuche deutlich einschränken. Nimmt man an, dass das Environment gemäß **M360** definiert ist, und dieses nicht mehr manipuliert werden kann, dann ist folgende Reihenfolge sinnvoll: `E` - Environment, `P` - POST, `S` - Server für `variables_order`:

```
register_globals   = Off
variables_order    = "EPS"
```

(3) Definieren der Verzeichnisse für Includes und Uploads

Einige PHP-Funktionen können Dateien aus beliebigen, auch nicht-lokalen Quellen einbinden. Diese Möglichkeit zur Einbindung nicht-lokaler Dateien sollte mit `allow_url_fopen=Off` deaktiviert werden. Ebenso sollte festgelegt werden, aus welchen Verzeichnissen Include-Dateien eingebunden werden dürfen, bzw. in welchen Verzeichnissen Daten abgelegt werden dürfen.

```
allow_url_fopen       = Off
safe_mode_include_dir = /ServerRoot/PHP-includes/
upload_tmp_dir        = /DocumentRoot/uploads/tmp/
```

(4) Ausführbare externe Programme

Von PHP-Skripten können beliebige weitere PHP-Skripte oder Programme aufgerufen werden. Um Manipulationen vorzubeugen, sollten die zur eigentlichen Applikation gehörenden Programme nur aus einem definierten Verzeichnis gelesen werden können:

```
open_basedir       = /ProgramRoot/execphp/
safe_mode          = On
safe_mode_exec_dir = /ProgramRoot/execphp
```

Zusätzlich können bestimmte Funktionen deaktiviert werden. Insbesondere `phpinfo()` und `show_source()` sind auf einem produktiven System normalerweise nicht erforderlich:

```
disable_functions     = system, exec, shell_exec, passthru,
                        phpinfo, show_source, popen, proc_open
```

(5) Sessions absichern

In der Grundeinstellung verwaltet PHP die Sessiondaten in Dateien, deren Name die SessionID ist. Gespeichert werden diese Dateien unter `/tmp` in UNIX bzw. `\temp` unter Windows. Üblicherweise ist das Verzeichnis unter `/tmp` ungeschützt (Verzeichnis für jedermann schreibbar, Dateien gegebenenfalls für jederman lesbar). PHP selbst vergibt zwar sehr restriktive Zugriffsrechte für diese Dateien (nur der Eigentümer darf lesen und schreiben), was aber in der Praxis unzureichend ist, da jeder Zugriff via Webserver genau mit diesem Benutzer erfolgt. Um das Risiko der Datenmanipulation und/oder Datenausspähung zu minimieren, muss dieses Verzeichnis geändert werden. Außerdem ist die Zeit, für die eine Session gültig ist, zu reduzieren, etwa 1 Stunde (Standard ist 1 Tag):

```
[sessions]
  session.save_path    = /ProgramRoot/conf
  session.save_handler = files
  session.gc_maxlifetime= 60
```

Sessions können weiter abgesichert werden indem man den Cookie-Namen ändert, die Cookies restriktiver setzt (Host, Path) und die Gültigkeitsdauer der Cookies limitiert:

```
session.name            = CookieName
session.cookie_secure   = On
session.cookie_lifetime = 3600
session.use_trans_sid   = On

session.cookie_path     = /sicher-anwendung/
```

```
session.cookie_domain   = www.unternehmen.tld

session.cache_expire    = 60
session.cache_limiter   = nocache
```

(6) Datenbankanbindung

Wird mit PHP auf eine Datenbank zugegriffen, dann sind auch diese Einstellungen sicher zu konfigurieren:

```
[PHP]
  mysql.default_host   = sqlhost
  mysql.default_port   = 3306
  mysql.default_socket = /pfad/zu/mysqlsock
  mysql.default_user   = wwwsql
  mysql.default_password = sicher
  mysql.trace_mode     = Off
```

(7) Eingabedaten maskieren

Die Einstellung `magic_quotes_gpc=On` bewirkt, dass in der PHP-Umgebung alle problematischen Anführungszeichen und Hochkomma mit einem vorangestellten Backslash versehen werden:

```
[PHP]
  magic_quotes_gpc     = On
```

Achtung: diese Einstellung schützt nicht vor XSS und SQL-Injection, sie schützt nur das PHP-Skript selbst.

Bemerkungen

Allgemein gilt: alle Werte (Pfadnamen, Timeout) in obigen Beispielen sind dem Einsatz entsprechend anzupassen. Die obigen Beispiele beziehen sich auf eine einzelne `php.ini`, welche für genau einen Webserver ohne "virtual Hosts" benutzt wird. Insbesondere in Virtual-Host-Umgebungen (z.B. in der Kombination Apache und PHP) sind die Einstellungen in `php.ini` nicht ausreichend. Für Apache mit `mod_php` bietet sich dafür z.B. die `php_admin_value`[5] Directive in `httpd.conf` an.

(2) In der Anwendung sollte geprüft werden, ob für `variables_order` zusätzlich auch noch `G` - GET und/oder `C` - Cookie gesetzt wird.

(4) Das alleinige Setzen von `safe_mode=On` ist aufgrund bekannter Probleme in der Implementierung von PHP nicht ausreichend. Außerdem kann `safe_mode=on` zu weiteren Problemen mit den Dateiberechtigungen führen (z.B. wenn der Besitzer der Datei nach einem FTP-Upload nicht der des Webservers ist). `safe_mode` sollte also immer zusätzlich mit `open_basedir` abgesichert werden. Ebenso sollte man in `safe_mode_exec_dir` keine allgemeinen Pfade wie z.B. `/usr/bin` eintragen. In dem durch `safe_mode_exec_dir` referenzierten Verzeichnis dürfen wirklich nur genau die Programme liegen, die von der Webanwendung gebraucht werden.

(5) PHP speichert Session-Daten zunächst in normalen Dateien. Alternativ kann PHP auch andere Methoden dafür verwenden. Hierfür ist in `php.ini` die Einstellung `session.save_handler=user` zuständig. Details dazu, z.B. zur Verwendung einer SQL-Datenbank, sind in der Dokumentation von PHP zu finden.

Die zusätzlichen Schritte müssen mit den Maßnahmen **M170,** insbesondere **M170.2**, **M170.3**, **M170.5** und **M170.7**, abgestimmt werden und stellen hier nur beispielhafte

[5] Achtung: php_admin_value kann nicht alle Konfigurationsvariablen besetzen

Empfehlungen dar.

(6) Gelegentlich wird empfohlen, die für die Datenbankanmeldung notwendigen Anmeldedaten (Benutzername und Passwort) nicht in den PHP-Skripten selbst, sondern in den Umgebungsvariablen des Webservers zu hinterlegen (z.B. PHP Cookbook). Beide Methoden haben Vor- und Nachteile. Die Umgebungsvariablen sind z.B. mit `phpinfo()` sichtbar, die Daten im PHP-Skript selbst könnten gegebenenfalls über andere Schwachstellen erreicht werden. Die Hinterlegung der Anmeldedaten (`mysql.default_user`, `mysql.default_password`) in `php.ini` ist daher ein guter Kompromiss, wenn man die anderen Maßnahmen bzgl. der sicheren Konfiguration dieser Datei (z.B. **M350**) beachtet.

Prinzipiell vorteilhaft wäre die Speicherung der Anmeldedaten in der Konfigurationsdatei des Webservers (z.B. `http.conf`), die nur von `root/administrator` lesbar sein sollte. Der Webserver würde dann eine persistente Verbindung zum Datenbankserver aufbauen. Allerdings ist für PHP keine entsprechende Lösung bekannt. Man kann jedoch mit der Directive `php_admin_value` in der `httpd.conf` bestimmte PHP-Konfigurationsvariablen belegen.

(7) Diese Einstellung entbindet nicht davon, auf *Data Validation* (siehe **M100** bis **M150**) zu achten.

2.32 M400 Serverkonfiguration: Web Application Isolation

Abstract **Auf jedem Rechner sollte genau ein Web- oder Application- oder Datenbankserver laufen.**

Ebene **System**

Beschreibung

Jede Anwendung sollte unter ihrer eigenen Benutzerkennung laufen (vgl. **M340)**. Trotzdem kann eine Schwachstelle in nur einer Anwendung alle anderen Anwendungen auf dem System mit gefährden.

Wird hingehen jede Webanwendung auf einem eigenen Rechner realisiert, kann man von *Web Application Isolation* sprechen.

Beispiele

Eine Webanwendung bestehend aus Web-, Application- und Datenbankserver wird auf 3 Rechner verteilt. Die einzelnen Konfigurationen unterscheiden sich nicht von denen in den Maßnahmen **M370, M340**, **M350**, **M380**, **M390**. Weitere Beispielkonfigurationen finden sich in [16] bis [20].

Bemerkung

Die erhöhte Sicherheit wird durch einen erhöhten Aufwand für Konfiguration der einzelnen Rechner und evtl. zusätzliche Konfiguration auf der Firewall erkauft. Im konkreten Anwendungsfall sind die zusätzlichen Aufwände in Relation zum Sicherheitsgewinn zu setzen.

2.33 M405 Vermeidung einer zu hohen Fehlertoleranz

Abstract **Vermeidung einer zu hohen Fehlertoleranz**

Ebene **Implementierung**

Beschreibung

Webentwickler tendieren dazu, Anwendungen zu tolerant gegenüber Fehlern auszulegen. So z.B. wird ein Programm mit Zusatzfunktionen zur Behandlung von Fehlerzuständen ausgestattet, obwohl die Ursache dieser Fehlerzustände nicht vollständig geklärt ist. Zu den prinzipiell möglichen Fehlerursachen zählen sowohl eigene Programmierfehler, als auch jene Probleme, die aus dem oft komplexen Zusammenspiel von Server, Browser und Netzwerkarchitektur entstehen. Fehlerzustände können aber auch aus absichtlich manipulierten Eingabedaten resultieren (vgl. **M120**).

Eine korrekt programmierte Webanwendung sollte robust und fehlertolerant gegenüber bestimmungsgemäßer Bedienung sein, die Session aber invalidieren, wenn ein offensichtlicher oder vermuteter Missbrauchsversuch vorliegt.

2.34 M410 Verschiedene Maßnahmen

Abstract **Hier werden verschiedene Einzelmaßnahmen und Best Practices zusammengefasst.**

Beschreibung

M410.1 Signierte Werte

Wenn die Anwendung Werte an den Browser sendet, die dieser nicht verändern darf (z.B. in Auswahllisten, SessionID), die aber mit einem Form-Request wieder zur Anwendung zurückkommen, dann muss die Anwendung die erhaltenen Werte mit *Data Validation* auf gültige Werte prüfen.

Sollen die eigentlichen Werte zusätzlich vor Ausspähen geschützt werden, so empfiehlt sich die Verwendung von signierten Hashes anstatt der "Klartext"-Werte.

Die folgenden Beispiele zeigen, wie man z.B. eine SessionID als `Hidden`-Field transportiert[6].

Beispiel in Perl

Variable erzeugen und in die Ausgabe schreiben:

```
$id   = "grosse Zufallszahl darf nicht manipuliert werden";
$salt = "und noch ein Geheimnis dazu";
$hash = Digest::MD5::md5_base64($salt.$id);
print '<input type="hidden" name="sid" value="'.$hash.'" />';
```

Variable eines Requests prüfen:

```
$id   = "grosse Zufallszahl darf nicht manipuliert werden";
$salt = "und noch ein Geheimnis dazu";
$hash = md5($salt.$id);
$sid  = validateInput($q->param('sid');
if ($hash == $sid) {
   # weiter
} else {
   print 'Daten manipuliert.';
   exit(1);
}
```

Beispiel in PHP

Variable erzeugen und in die Ausgabe schreiben:

```
$id   = "grosse Zufallszahl darf nicht manipuliert werden";
$salt = "und noch ein Geheimnis dazu";
$hash = md5($salt.$id);
print '<input type="hidden" name="sid" value="'.$hash.'" />';
```

Variable eines Requests prüfen:

```
$id   = "grosse Zufallszahl darf nicht manipuliert werden";
$salt = "und noch ein Geheimnis dazu";
$hash = md5($salt.$id);
$sid  = validateInput($_REQUEST['sid']);
```

[6] Eine erforderliche Input Data Validation ist durch validateInput() hier nur angedeutet

```
if ($hash == $sid) {
   // weiter
} else {
   print 'Daten manipuliert.';
   exit(1);
}
```

M410.2 Perl tainted mode

Alle Perl CGIs sollten im tainted mode (Option -T) betrieben werden. Der tainted mode ist zwar kein genereller Schutz vor gefährlichen Parametern und Zeichen, aber Perl markiert damit zur Laufzeit des CGI-Skripts alle Parameter, die an Systemfunktionen übergeben werden und erzwingt, dass diese explizit geprüft werden müssen.

In Apache mit `mod_perl` kann der tainted mode auch standardmäßig in der `httpd.conf` eingestellt werden:

```
PerlWarn         On
PerlTaintCheck   On
PerlModule       strict
```

M410.3 Server Side Includes in Apache absichern

Wir verweisen auf die Quelle [21] "Apache Tutorial: Introduction to Server Side Includes".

M410.4 Sicherheitskritische Bereiche zusätzlich absichern

Bevor einem bereits eingeloggten Benutzer Zugang zu besonders vertraulichen Informationen oder sicherheitskritischen Bereichen einer Webanwendung gewährt wird, sollte eine erneute Passwort-Abfrage stattfinden. Dadurch kann eine gleichzeitige Kompromittierung des sicherheitskritischen Bereiches verhindert werden.

M410.5 Vertrauliche Informationen nicht anzeigen

Zusätzlich zu **M410.4** sollte die Webanwendung Vorkehrungen für den Fall einer Kompromittierung treffen. Sensible Informationen wie Passwörter sollten daher generell nicht angezeigt werden.

2.35 M420 Einsatz von Web Application Security Tools

Abstract **Neben dem Einsatz von WebShields als Application-Firewalls sollten insbesondere auch Web-Scanner zum gezielten Aufdecken bestehender Schwachstellen in Webanwendungen zur Anwendung kommen.**

Ebene **System**

Beschreibung

2.35.1 WebShields

Während auf Netzwerk-Ebene mittlerweile ausgereifte Firewall-Technologien zur Verfügung stehen, gab es auf der Ebene der Webanwendungen lange Zeit keinen vergleichbaren Schutz. Die jetzt existierenden WebShield-Produkte haben ihre erste Erprobungsphase nun hinter sich.

WebShields können auch als *Web Application Firewall* bezeichnet werden. Sie filtern den Datenstrom zwischen Browser und Webanwendung. Tritt ein als unzulässig eingestuftes Eingabemuster auf, wird der Transfer unterbrochen oder auf eine andere, vorher festgelegte Weise reagiert. Ein WebShield arbeitet als Proxy im Datenstrom, kennt somit das Protokoll der Anwendung.

Grundsätzlich filtern WebShields die vom Browser an den Webserver übertragenen Daten. Einige WebShield-Produkte sind zusätzlich in der Lage, die vom Webserver an den Browser versandten Daten zu überwachen. Hierbei kann das WebShield "lernen", wie die Daten beschaffen sind. Dadurch wird ein späterer Vergleich zwischen aktuellen und gelernten Daten ermöglicht. So können Filter in eingeschränktem Maße verhindern, dass der Browser schadhaften Code erhält (z.B. von einer Webanwendung, die keine ausreichende Output-*Data Validation*, siehe **M100**, durchführt).

WebShields sind nicht in der Lage, alle Angriffsformen auf Webanwendungen zu erkennen. Generell gilt: Sicherheitsprobleme sollten in der Webanwendung selbst gelöst werden. WebShields sollten als zusätzliche Schutzmaßnahme in Betracht gezogen werden insbesondere für folgende Fälle:

- In vielen Unternehmen werden Webanwendungen von einer Vielzahl externer Partner entwickelt. Die Sicherheit liegt in den Händen der externen Partner.
- Bei bestehenden Anwendungen ist ein nachträgliches Security-Review bzw. ein Beheben von Sicherheitsproblemen in der Anwendung häufig gar nicht möglich, sei es aus Kostengründen oder weil die Entwickler nicht mehr verfügbar sind.
- Die Absicherung über mehrere Sicherheitsstufen ist ein wesentliches Prinzip für höhere Sicherheit (Second Line of Defense). WebShields leisten dabei gute Dienste.

WebShields in der IT Infrastruktur

WebShields sind Komponenten in der IT-Infrastruktur, die weniger leicht als beispielsweise Netzwerk-Firewalls zu integrieren sind. Die Lastanforderungen (Performance-Einbußen durch permanente Prüfung des Datenverkehrs) und die Abhängigkeiten zu anderen Systemen, wie Loadbalancern, Proxys und Cluster-Architekturen, sind deutlich größer.

Die aktuell verfügbaren Produkte weisen einen unterschiedlichen Funktionsumfang auf. Dieser reicht vom einfachen Filtern unzulässiger Zeichen bis hin zu komplexen Systemen, die versuchen, die Anwendungslogik (zumindest teilweise) zu verstehen. Generell unterscheiden lassen sich Software-WebShields, Appliances, Webserver-spezifische Produkte, und sonstige. Während ein Software-WebShield als reines Softwareprodukt auf vorhandener Hardware zzgl. Betriebssystem installiert werden muss, bezeichnet eine WebShield-Appliance eine als Produkt erhältliche Kombination aus Hard- und Software. Zu den Webserver-spezifischen Produkten zählen auch die frei verfügbaren Tools "mod_security" und "BruteWatch":

mod_security

Eine besondere Stellung unter den Filtern nimmt das frei verfügbare *mod_security* für Apache ein. *mod_security* ist ein Input-Filter, der mit Regeln ausgestattet werden kann. So kann mod_security z.B. XSS, SQL-Injection, Null-Byte oder Path Traversal erkennen und entsprechend reagieren. Mit den "Advanced Filtering" genannten Techniken kann selektiv auf bestimmte URLs oder auf bestimmte Werte im HTTP-Header reagiert werden. Es ist möglich, eigene, externe Programme aufzurufen, womit die Funktionalität fast beliebig erweitert werden kann.

Eine beispielhafte Konfiguration in `httpd.conf`, die den oben genannten Funktionsumfang hat, wäre:

```
SecFilterEngine           On
SecFilterCheckURLEncoding On
SecFilterForceByteRange   32 126
SecAuditEngine            RelevantOnly
SecAuditLog               /ServerRoot/logs/secaudit_log
SecFilterScanPOST         On
SecFilterDefaultAction    "deny,log,status:406
SecFilter                 "\.\./"
SecFilter                 "/\.$"
SecFilter                 "<(.|\n)+>"
SecFilter                 "\"[[:space:]]*>"
SecFilter                 "'[[:space:]]*>"
SecFilterSelective        "HTTP_CONTENT_TYPE" multipart/form-data
```

Perl Module Apache::BruteWatch

Das Perl-Modul *Apache::BruteWatch* überwacht die Authentisierung des Webservers (Basic Authentication). Es greift ein, wenn mit Brute-Force-Angriffen versucht wird, Passwörter herauszufinden. Apache selbst (bis einschl. Version 2.0.50) hat keine eigene Möglichkeit, dies zu verhindern.

Bei Verwendung von Apache::BruteWatch ist zunächst eine Tabelle in einer Datenbank anzulegen, auf die Apache dann Zugriff hat, z.B. für MySQL:

```
CREATE TABLE bruteattempt (
  id       INT(11)      NOT NULL auto_increment,
  ts       INT(11)      DEFAULT NULL,
  username VARCHAR(255) DEFAULT NULL,
  PRIMARY  KEY (id)
) TYPE=MyISAM;

CREATE TABLE brutenotified (
  id       INT(11)      NOT NULL auto_increment,
  username VARCHAR(255) DEFAULT NULL,
  ts       INT(11)      DEFAULT NULL,
  PRIMARY  KEY (id)
) TYPE=MyISAM;
INSERT INTO db VALUES ('FQDN', 'brutelog', 'user',
                       'Y','Y','Y','Y','Y','N','N','N','N','N');
```

Dann wird Apache::BruteWatch in der `httpd.conf` konfiguriert:

```
PerlLogHandler                Apache::BruteWatch
PerlSetVar BruteDatabase      DBI:mysql:brutelog
PerlSetVar BruteDataUser      user
PerlSetVar BruteDataPassword  password
PerlSetVar BruteMaxTries      5
PerlSetVar BruteMaxTime       120
PerlSetVar BruteForgive       84600
PerlSetVar BruteNotify        admin@server.tld
```

2.35.2 Web Scanner

*Web Scanner (*oder: *Web Application Scanner)* sind Programme für Penetrationstests. Sie untersuchen eine Anwendung auf bekannte Schwachstellen und unterstützen dabei den Webentwickler oder Penetrations-Tester beim Auffinden von Sicherheitslücken.

Funktionsweise

Ein Web Scanner wird auf einem Client-Computer installiert, um dann die Webanwendung zu analysieren. Im Zuge der Analyse können folgende vier Phasen durchlaufen werden:

Crawl/Scan

Ein Web Scanner bewegt sich ähnlich wie eine Suchmaschine durch die gesamte Webanwendung. Dabei werden sämtliche Links besucht und gegebenenfalls Form-Felder mit Testwerten ausgefüllt. Der Scanner erhält so ein umfassendes Wissen über den Aufbau der Webanwendung, die Funktionsweise von Dialogschritten und den Inhalt von Formular-Seiten.

Die Webanwendung wird auf häufig vorhandene Installations-, Konfigurations- oder Testverzeichnisse, sowie bekannte problematische oder informative Dateien durchsucht (Stichwörter: Directory Enumeration, File Enumeration, Directory Traversal, Forceful Browsing).

Unterstützt werden kann die Scan-Phase durch Einbeziehen großer öffentlicher Suchmaschinen. Vorausgesetzt wird hierfür, dass die eigene Website bereits durch diese Suchmaschinen indexiert worden ist. Die Ergebnisse dieses Schrittes wurden dann auf den Suchmaschinen gespeichert (Index/Cache). Im Rahmen einer Scan-Phase können diese Suchmaschinen nun gezielt nach Informationen über die eigene Website abgefragt werden. Für die Zusammenstellung von Suchmustern, sowie Automatisierung der Abfragen kann die Nutzung spezieller Tools zweckmäßig sein.

Analyse

In einem nächsten Schritt wird die Webanwendung einer eingehenden Sicherheitsanalyse unterzogen. Die gewonnenen Erkenntnisse können in einer Datenbank hinterlegt werden: Typ und Version des Webservers, verwendete Technologien und Tools (CGI, Servlets, JSP, JavaScript, PHP, usw.), Verwendung von Cookies oder anderer Mechanismen zum Session-Tracking, Analyse der Form-Parameter einer Seite, insbesondere auch die Verwendung von `Hidden`-Parametern, Extraktion von Kommentaren.

Audit/Penetrationstest

Unter Einbeziehung des in der vorangegangenen Phase gewonnenen Wissens werden

systematisch fehlerhafte oder unzulässige Eingabemuster erzeugt, und an die Webanwendung versandt. Einige der existierenden Scanner unterteilen diese Phase in eine schadlose, und eine potentiell schadhafte Prüfung.

Reporting

Die Ergebnisse der Scan- und Audit-Phase werden in Reports zusammengefasst. Der Umfang reicht dabei von Executive Summeries mit Nennung nur der größten Schwachstellen bis hin zu detaillierten Beschreibungen, in denen auch erste Hinweise für die Behebung des jeweiligen Problems gegeben werden können.

Der Nutzen von Web Scannern

Allgemein gilt: Web Scanner sind sinnvolle Tools, um Experten bei der Analyse der Sicherheitseigenschaften einer Webanwendung zu unterstützen. Sie sind jedoch weder in der Lage, fehlendes Know-how zu kompensieren, noch können sie Experten ersetzen. Die einer Webanwendung innewohnende Komplexität und die Tatsache, dass manche Schwachstellen von einem Tool grundsätzlich nicht erkennbar sind, haben zur Folge, dass weiterhin geübte Personen für den Großteil der Analyse benötigt werden. Eine Interpretation eines Web-Scan-Reports ist ohne Erfahrung im Testen von Webanwendungen nur schwer möglich.

3 Anhänge

3.1 Referenzliste

Trotz sorgfältiger Prüfung kann das BSI für die hier verlinkten Inhalte keine Haftung übernehmen.

[1] http://www.owasp.org/index.php/Category:OWASP_Guide_Project
OWASP – A Guide to Building Secure Web Applications Version 2.0

[2] http://www.owasp.org/index.php/Category:OWASP_Guide_Project
OWASP Guide to Building Secure Web Applications and Web Services Version 3.0 Working Draft

[3] http://sourceforge.net/project/shownotes.php?release_id=230454
OWASP Web Application Penetration Checklist

[4] http://www.owasp.org/index.php/Category:OWASP_Project
The OWASP Testing Project

[5] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tsqlref/ts_xp_aa-sz_4jxo.asp
xp_cmdshell deaktivieren

[6] http://rotanovs.com/php-session-httponly.patch
Patch zur Unterstüzung des HttpOnly-Flags in PHP4

[7] http://msdn.microsoft.com/workshop/author/dhtml/httponly_cookies.asp
Protecting Data with HTTP-only Cookies

[8] http://www.rfc.net/rfc2965.txt
"HTTP State Management Mechanism" – Version 1-Cookies

[9] http://www.technicalinfo.net/papers/WebBasedSessionManagement.html
Web Based Session Management

[10] http://www.w3.org/2001/tag/doc/whenToUseGet.html
GET vs. POST

[11] http://www.captcha.net/
The Captcha Project

[12] http://www.codeproject.com/aspnet/CaptchaImage.asp
Captcha-Implementierung für ASP.

[13] http://www.bsi.bund.de/literat/studien/sistudien/Apache_2003.pdf
Apache Weberver Sicherheitsstudie

[14] http://www.bsi.bund.de/literat/studien/sistudien/IIS_2003.pdf
Microsoft Internet Information Server Sicherheitsstudie

[15] http://windowsecurity.com/articles/MSSQL-Security.html
sicheres MS-SQL Setup

[16] http://java.sun.com/j2ee/1.4/docs/api/javax/servlet/http/HttpServletRequest.html

[17] http://livedocs.macromedia.com/coldfusion/6.1/htmldocs/clusteri.htm
Application Isolation für ColdFusion J2EE

[18] http://livedocs.macromedia.com/coldfusion/6.1/htmldocs/cluster8.htm
Application Isolation für ColdFusion J2EE im Apache

[19] http://www.microsoft.com/windows2000/en/server/iis/htm/core/iiwarndc.htm#app_2
Application Isolation für IIS 5

[20] http://www.microsoft.com/technet/prodtechnol/windowsserver2003/technologies/webapp/iis/appisoa.mspx
Application Isolation für IIS 6

[21] http://www.apache.org/docs/2.2/howto/ssi.html
Apache Tutorial: Introduction to Server Side Includes

3.2 Glossar

%-Encoding	Synonym für URL-Encoding. Kodierung von Sonderzeichen in der URL mit %xx.
Adaptive Proxy	früher benutzter Begriff für Application Level Firewall
Anwender	Person, die ein Programm oder eine Applikation benutzt..
Application Level Firewall	Firewall die den Datenstrom auf der Applikationsebene (OSI Level 7) kontrolliert; siehe auch WebShield
Applicationserver	spezieller Dienst/Programm, welches komplexe Daten zur Verfügung stellt
Bedrohung	potenzielle Gefahr für Daten oder Systeme
Benutzer	Person, die ein Programm, Applikation benutzt. Auch: Name einer Benutzerkennung des Systems (z. B. aus /etc/passwd)
Benutzerkennung	siehe Benutzer
Blacklist	Liste mit unerlaubten Zeichen, Wörter, etc.
Brute Force	Angriffstechnik, bei der mittels Durchprobieren aller Möglichkeiten Passwörter etc. ermittelt werden
Buffer Overflow	spezielle Angriffstechnik auf Variablen (memory)
Cache	"flüchtiger Zwischenspeicher" (zeitlich begrenzt)
Code-Injection	einschleusen von HTML-Code durch die verwendete Skriptsprache auf dem Server, z. B. PHP (Variante von HTML-Injection)
Command-Injection	spezielle Angriffstechnik mit dem Ziel, Kommandos im System auszuführen
Content Branding	einheitliches Aussehen des Inhaltes
Cookie-Injection	spezielle Angriffstechnik auf Cookies
Cookie Poisoning	Manipulation des Wertes eines Cookie
Cookie Tampering	siehe Cookie-Manipulation
Countermeasure	Gegenmaßnahme
Cracker	Person, die mit unerwünschten, kriminellen Mitteln versucht, auf einem Computer Daten zu manipulieren oder zu stehlen
Credentials	Authentifizierungsdaten
Cross-Site Scripting	Manipulation von Parametern, so dass im Browser Skriptcode ausgeführt wird; häufig auch synonym zu HTML-Injection verwendet

Database Gesamtheit der Daten in einer Datenbank

Data Validation Prüfung der Daten in Bezug auf Web Application Security

Defacement Manipulation des Inhaltes einer Website

Denial of Service Überflutung / Die Webanwendung wird funktionsunfähig gemacht

Directory Enumeration Erraten von Verzeichnisnamen durch systematisches Probieren

Directory Traversal Zugriff auf Verzeichnisse außerhalb des DocumentRoot

DocumentRoot Verzeichnis, in dem sich die Dokumente eines Webservers befinden

DoS siehe: Denial of Service

Encoding Kodierung, im Gegensatz zu Encryption = Verschlüsselung (eine Kodierung ist i. A. direkt umkehrbar)

Encryption Verschlüsselung, im Gegensatz zu Encoding = Kodierung (eine Verschlüsselung ist nur mit einem Schlüssel zu entschlüsseln)

Enumeration Aufzählung, Nummerierung (z. B. von Dateien), systematisches Probieren. Siehe auch 'Brute Force'

Environment Umgebungskontext, in dem ein Programm gestartet wird (insbesondere die Environmentvariablen)

Escape-Encoding Synonym für %-Encoding

Exploit konkrete Ausnutzung einer Schwachstelle, zumeist in Form eines kleinen Programms oder Skripts, das mit wenig Vorkenntnissen genutzt werden kann

Forceful Browsing Direkter Zugriff auf eine URL (unter Umgehung der Applikationslogik)

Gefährdung tatsächlich bestehende Möglichkeit, dass Schaden entsteht

Hacker Person, die Schadfunktionen, Gefahren und Bedrohungen aufzeigt, diese aber im Gegensatz zum Cracker nicht (kriminell) ausnutzt

Hidden Field-Manipulation Änderung von `Hidden`-Parametern

HIDDEN-Parameter spezieller Parameter in einer Webseite, welcher für den Anwender normalerweise nicht sicht-/änderbar ist

Horizontal Privilege Escalation Rechte eines anderen Benutzers oder Verzeichnisses können erlangt werden

HTML-Encoding Umwandlung eines Zeichens zum HTML-Entity

HTML-enkodiert String (Text), der ein HTML-Entity repräsentiert

HTML-Escape %-Encoding im HTML Text

HTML-Injection Einschleusen von HTML-Code (Variante von, aber im Gegensatz zu Cross-Site Scripting)

HTTP Response Splitting spezielle Angriffstechnik zur Manipulation des HTTP Header

Identity Theft Diebstahl von persönlichen Daten/Merkmalen (z. B. Passwort)

Information Disclosure Verletzung der Vertraulichkeit, Mithören, Ausspionieren

Information Gathering Informationsbeschaffung

Input Validation Data Validation bezogen auf Eingabedaten zur Webapplikation

Malicious Code jede Form von unerwünschten Daten

Malware Oberbegriff für jede Form von unerwünschter oder störender Software

Named Character Reference Synonym für HTML-Encoding mit benamten Symbolen (Terminologie des W3C)

Numeric Character References Synonym für HTML-Encoding mit numerischen Symbolen (Terminologie des W3C)

Output Sanitation Synonym für Output Validation

Output Validation Data Validation der Ausgabe der Daten zum Browser oder zu anderen Servern

Parameter-Manipulation Manipulation von Parametern (Name und/oder Wert)

Parameter Tampering siehe Parameter-Manipulation

Path Traversal Spezielle Angriffstechnik. Siehe Directory Traversal

Penetrationstest Überprüfung einer Applikation (insbesondere mit fehlerhaften Eingaben)

Percent-Encoding siehe %-Encoding

Predictable Resource Locations vorhersagbare Resourcen, siehe auch Directory Enumeration

Privilege Escalation fehlerhafte Erhöhung oder Verschiebung von Privilegien, Zugriffrechten

rc-Skript spezielles Skript zum Starten eines Dienstes

Referer spezielle Variable im HTTP-Header (die Schreibweise ist nicht korrekt, sie wird hier nur verwendet, wenn genau diese Variable gemeint ist, sonst siehe ‘Referrer’)

Referrer URL der Seite, von der aus verlinkt wurde

Sandbox spezielle, eingeschränkte Umgebung für Dienst/Programm/Skript

Scanner allgemeine Bezeichnung für ein Programm, welches Informationen durch systematische Tests sammelt; siehe auch Webscanner

ServerRoot Verzeichnis, in dem die Software für einen Server installiert ist

Session logische Verbindung zwischen Client und Server; kann über mehrere Einzel-Zugriffe hinweg bestehen

SessionID eindeutiger Bezeichner für eine Session (diese Schreibweise wird benutzt, wenn der Wert gemeint ist, der eine Session eindeutig identifiziert, z. B. in einem Cookie)

Server damit ist immer ein Computer mit installiertem Betriebssystem und evtl. installierten Programmen gemeint

Session Fixation spezielle Angriffsmethode

Session-Injection siehe Session Fixation

Shell Command-Injection siehe Command-Injection

(Web-)Shield Firewall auf Applikationsebene

Skript Programm, das in einer Interpretersprache geschrieben ist

Social Engineering Missbrauch eines Vertrauensverhältnisses zum Erschleichen von (persönlichen) Informationen

Spidering alle URLs einer Webapplikation aufsammeln

SQL-Injection spezielle Angriffsmethode auf Datenbanken

SSI-Injection spezielle Angriffsmethode auf dynamische Webseiten (mit SSI)

Stealth Commanding Synonym für Command-Injection

Unicode Standard für multilingualen Zeichensatz

URL-Encoding Umwandlung eines Zeichens als %-Encoding

URL-enkodiert String (Text) der ein Zeichen in %-Encoding repräsentiert

URL-Escape %-Encoding in URL

Vertical Privilege Escalation für den aktuellen Benutzer oder das aktuelle Verzeichnis können weitere Rechte erlangt werden

Vulnerability Verwundbarkeit, Angriffspunkt, Schwachstelle

War Googling siehe War Searching

War Searching automatisiertes Auffinden von Schwachstellen in Websites mit Suchmaschinen

Web Application Firewall siehe WebShield

Web Application Shield siehe WebShield

Webanwendung	Synonym für Webapplikation
Web Application Isolation	Trennung von Daten und Prozessen/Threads einer Webanwendung
Webapplikation	Programm oder Gruppe von Programmen, die zusammen eine Anwendung realisieren die mittels eines Webservers bedient wird
Webformular	Formular, das sich auf einer Webseite befindet
Web Scanner	Programm zum Testen einer Webapplikation
WebShield	Application Level Firewall, die einen Webserver schützt
Webserver	Dienst/Programm, welches via HTTP Daten zur Verfügung stellt
Webservice	Dienst, der mit XML-formatierten Meldungen über HTTP kommuniziert
Website	alle Seiten, die ein Webserver zu einer Domain liefert
Website-Spoofing	Fälschung einer Website oder Webseite
Webspace	Platz (auf Festplatte), den eine Website benötigt
Whitelist	Liste mit erlaubten Zeichen oder Zeichenketten
WS-Security	siehe Web Service Security
Web Service Security	Sicherheit für Web Services (XML, SOAP)
Xpath-Injection	spezielle Angriffsmethode auf Pfade in XML
XSS	Akronym für Cross-Site Scripting
Zugriffsrechte	Berechtigungen (Permissions) für Dateien, Verzeichnisse oder Prozesse