

# Sicherheit im Kontext von PHP und Webanwendungen

In diesem Kapitel gibt es einen Rückblick auf PHP als Sprache für Webanwendungen – und warum PHP so unschlagbar oft in Verbindung mit webbasierten Applikationen verwendet wird. Schließlich gibt es noch einen Ausflug in die Welt der Sicherheitskonzepte.

## 1.1 Historie: PHP

Wäre PHP seit jeher konsequent mit Fokus auf Sicherheit entwickelt worden, wäre dieses Buch entweder nicht so umfangreich oder es würde schlichtweg nicht erscheinen. PHP wurde jedoch ursprünglich lediglich als einfacher Form-Interpreter entwickelt – er sollte lediglich HTML-Formulare auswerten.

Erst mit der Zeit hat es sich ergeben, dass mehr daraus wurde. Doch selbst als PHP in C neu entwickelt wurde, war noch nicht absehbar, was aus dieser Sprache einmal werden würde – und vor allem konnte niemand ahnen, welche Sicherheitslücken und konzeptionellen Probleme sich mit der teilweise unkontrollierten Verbesserung des Interpreters ergeben würden.

Weiterhin kann man feststellen: PHP stammt noch aus einer Zeit, zu der man in Bezug auf das Netz der Netze euphorisch war: Es gab diejenigen, die nichts damit anzufangen wussten, und diejenigen, die voller Hoffnung waren, was mit der globalen Vernetzung alles möglich sein wird. Keinesfalls kam jemand auf die Idee, dass es einmal – respektive heute – Menschen geben wird, die versuchen werden, Sicherheitslücken in Software auszunutzen um anderen wirtschaftlichen Schaden anzufügen. Hätte man das damals gewusst, so wäre die Entwicklung von PHP bestimmt eine andere gewesen: Entweder hätte es PHP nie gegeben, oder es wäre so restriktiv, dass es auf Dauer niemand verwenden würde.

PHP stand einmal für »Personal Homepage Tools« und erschien am 8. Juni 1995, der damalige Entwickler war Rasmus Lerdorf. Seine Intention für die Entwicklung: Protokollierung der Zugriffe auf seinen Online-Lebenslauf. Er veröffentlichte danach noch PHP/FI (Forms Interpreter) Version 2.0 am 12. November 1997; danach gab es eine für damalige Verhältnisse interessante Wendung: Die Version 3.0 von PHP wurde nicht mehr von Rasmus Lerdorf sondern von Andi Gutmans und Zeev Suraski entwickelt.

Rasmus Lerdorf hat mit den beiden Entwicklern zusammengearbeitet, sogar der »Rewrite Call« kam ursprünglich von ihm. PHP/FI wurde somit offiziell eingestellt, PHP war der zukünftige Name, wobei auch die Bedeutung in ein Backronym verändert wurde: PHP: Hypertext Preprocessor. Die beiden neuen Entwickler handelten aus der Motivation heraus, eine Sprache zu schaffen, mit der es möglich ist, komplexe eCommerce-Anwendungen zu realisieren, die in der Folge natürlich hauptsächlich Daten dynamisch erzeugen sollten.

Diese neue Version löste somit PHP/FI Version 2.0 ab, das eigentlich stets nur als Beta-Version existierte – PHP 3.0 erschien also am 6. Juni 1998, wobei es vorher eine neunmonatige öffentliche Testphase gab. Diese Version kann man als Durchbruch für die Verbreitung von PHP bezeichnen. Weiterhin kam hinzu, dass der Apache-Webserver immer größere Verbreitung fand (was auch daran lag, dass Microsoft die Entwicklung des Internets etwas »verschlafen« hatte, und somit kein weit verbreitetes Konkurrenzprodukt existierte). Die Synthese zwischen PHP und diesem Webserver wurde mit PHP 3.0 massiv vergrößert, wobei allerdings auch sichergestellt werden sollte, dass PHP 3.0 ebenfalls mit anderen Systemen oder ganz ohne Webserver-Software arbeiten kann.

PHP 3.0 hatte allerdings keinen Funktionsumfang, der eine professionelle Nutzung rechtfertigen würde – zumindest aus heutiger Sicht, es war jedoch ein Grundstein, da es damals lediglich mit Perl ein vergleichbares System gab. PHP war und ist allerdings um einiges leichter zu erlernen; dies gilt besonders dann, wenn man bereits vorher Erfahrungen in den Programmiersprachen C und/oder Java gemacht hat, da die Syntax an diese beiden Sprachen angelehnt ist.

Andi Gutmans und Zeev Suraski gründeten die Zend Technologies Ltd., die sich auch heute noch für die Entwicklung von PHP verantwortlich zeichnet, und starteten die Implementation der Zend Engine in Version 1; diese sollte später der Kern von PHP 4.0 werden. Mit dieser neuen Version wurde explizit die Sicherheit bei der Verwendung von globalen Variablen verbessert, es wurde auch ein Session-Management eingeführt (das für Anwendungen, die Daten zwischen verschiedenen Aufrufen transportieren müssen, essentiell ist) und die Unterstützung für alternative Webserver wurde deutlich verbessert. Das Veröffentlichungsdatum von PHP 4.0 war der 22. Mai 2000.

Doch die Entwicklung von PHP blieb natürlich nicht stehen, es ging weiter – auch wenn eine neue Version deutlich länger auf sich warten ließ, als man das von Version 3 und 4 gewohnt war. Am 13. Juli 2004 wurde PHP 5.0 freigegeben. Erstmals ist damit objektorientiertes programmieren möglich (Version 3 und 4 haben OOP nur sehr rudimentär unterstützt), auch wenn viele Funktionen von PHP selbst immer noch prozedural aufgerufen werden. Mit der Objektorientiertheit kam auch die Einführung von Exceptions, die die Fehlerbehandlung deutlich verbessern. Eine weitere wichtige Neuerung in Bezug auf das Web 2.0: DOM wurde objektorientiert realisiert, so ist es viel effizienter möglich, auf die Elemente eines DOM-

Baumes – wie er etwa bei der XML-Kommunikation mit einer AJAX-Client-Anwendung entsteht – zuzugreifen.

Die neueste Version, die einen Meilenstein darstellt, erschien etwa anderthalb Jahre später (24. November 2005): PHP 5.1. Hier kam nun mit PDO eine Datenbankabstraktionsschicht hinzu, mit der es möglich ist, mit den gleichen Funktionsaufrufen auf verschiedene SQL-basierte Datenbanken zuzugreifen.

## 1.2 PHP heute

PHP wurde über die Jahre weiterentwickelt und entspricht heute einer Interpretersprache, die als typischer klassifiziert werden kann.

### Hinweis

Typischer gilt eine Sprache dann, wenn der Compiler bzw. Interpreter vor Verarbeitung einen Test daraufhin vornimmt, ob die gewünschte Aktion mit den übergebenen Variablentypen zulässig ist. Im Fall von PHP wird zusätzlich noch versucht, die Daten in einen passenden Typ zu konvertieren.

Die Sprache an sich ist heute relativ mächtig; das liegt sowohl am vorhandenen Funktionsumfang als auch an der Möglichkeit, diese Funktionalität durch Module noch erweitern zu können. Beides – Komplexität und Modularität – tragen pragmatisch gesehen nicht sehr viel zur Sicherheit bei; auch war der Fokus bei PHP nicht auf ein extrem sicheres System gerichtet: Es sollte eine einfach erlernbare Sprache werden, die dennoch vielfältigste Aufgaben bei der Verarbeitung von Daten bewältigen kann.

Diese Einfachheit hat vor allem beim Umgang mit Variablen seine Spuren hinterlassen, die es so in vielen anderen Programmiersprachen nicht gibt. Die Möglichkeit, recht lax mit Variablen und bereitgestellten Daten umzugehen, hat allerdings auch im Hinblick auf Teile des PHP-Clientels – Menschen, die erst mit PHP den Einstieg in die Programmierung geschafft haben – zu einer Menge Problemen geführt. So werden Variablen angesprochen, ohne vorher sicherzustellen, ob diese überhaupt existieren oder gar gültige Werte enthalten.

Doch auch die universelle Einsetzbarkeit auf vielen verschiedenen Betriebssystemen hat zu teilweise starken Kompromissen bei der Sicherheit von beispielsweise temporären Dateien geführt. Diese Liste der Probleme, die durch das Konzept von PHP entstehen, eine Sprache zu schaffen, mit der es in kurzer Zeit möglich sein soll, komplexe eCommerce-Anwendungen zu erstellen, bedingt auf der anderen Seite Stolperfallen, die man umfahren muss, wenn man von der Sicherheit der Daten und der Applikation abhängig ist.

Und noch einmal: Auch wenn es so klingt, PHP ist keineswegs eine unsichere Sprache, die besser nicht verwendet werden sollte. Es handelt sich um ein System, mit dem viele komplexe Sachverhalte im Vergleich zu anderen Sprachen vergleichsweise einfach und dennoch effektiv abgebildet werden können. Diese übertriebene Negativbewertung von PHP auf diesen Seiten soll lediglich die Sensitivität für Softwaresicherheit besonders in Bezug auf Online-Applikationen erhöhen, denn dies ist das wichtigste um eine sichere Umgebung zu schaffen: Die Probleme müssen erkannt werden, bevor es möglich ist, sie zu beheben.

### 1.3 PHP und Apache

Schon seit PHP 3.0 gibt es eine starke Symbiose zwischen PHP und dem Apache-Webserver, der ebenfalls in seiner Umgebung eines der am meisten verbreiteten Systeme ist.

Die Integration von PHP in Apache ist denkbar einfach, denn neben der PHP-Konsolenanwendung kann – sofern der Apache bzw. dessen Tool `apxs` vorhanden ist und beim Kompilierungsvorgang berücksichtigt wird – gleich das Apache-Module erstellt und in die Konfiguration des Webservers eingebunden werden.

Weiterhin unterstützt der Apache nach dem Laden dieses Moduls auch einige neue Konfigurationsdirektiven, die Auswirkung auf die Funktionsweise von PHP haben und somit die PHP-seitige Konfiguration überschreiben.

Diese sehr direkte Kopplung kann allerdings auch ziemliche Probleme bereiten: Ist PHP als Apache-Modul aktiviert, ist der jeweilige Apache-Prozess, auf dem gerade eine PHP-Datei verarbeitet wird, stark abhängig von der erfolgreichen Arbeitsweise des PHP-Interpreters. Kommt es in diesem Fall zu einem Fehler oder gar dem GAU – einem Deadlock – so wird der entsprechende Apache-Client-Prozess nicht mehr korrekt reagieren. Im harmloseren Fall wird der Prozess einfach serverseitig abgebrochen und der Benutzer erhält den HTTP-Fehler 500. Im wesentlich schlechteren Fall – etwa wenn ein Deadlock auftritt – wird der Apache-Prozess nicht mehr reagieren; der Client erhält irgendwann ein Verbindungs-Timeout, danach wird der Apache-Prozess zwar weiterhin auf dem Server laufen und nicht mehr verwendbar sein, aber er wird wahrscheinlich nur etwas Arbeitsspeicher verbrauchen (es gibt auch hier Ausnahmen: kritisch wird es, wenn der Prozess auch noch Rechenzeit verbraucht). Allerdings kann es in seltenen Fällen auch zu einer gewissen Dramatik kommen: So ein hängender Prozess kann einen Server so in Mitleidenschaft ziehen, dass ein Abbrechen nicht mehr möglich und eventuell ein harter Reboot (hart bedeutet Power off!) notwendig ist. Es gibt ein Szenario, bei dem dieser Fall definitiv und relativ einfach reproduzierbar auftritt: Beim Lesen von großen Dateien, beachten sie hierzu auch unbedingt Kapitel 8 *Dateisystemzugriffe*.

Dies waren die Nachteile der starken Kopplung von Apache und PHP – jedoch kann man das ganze auch in das Gegenteil verkehren: Durch eine gut durchdachte Kom-

bination der Konfigurationen beider Systeme ist schon einmal ein gutes Stück Arbeit auf dem Weg zum sicheren System geschafft. Zudem kann man einen zusätzlichen Puffer schaffen und so die direkte Abhängigkeit des Webservers von der Ausführung des PHP-Interpreters aufheben, indem man PHP nicht direkt als Apache-Modul lädt, sondern die Ausführung etwa als CGI oder mittels suExec und ähnlichen Modulen ermöglicht. Wenn Sie sich dafür interessieren, so wird sicherlich Kapitel 11 das Richtige für Sie sein.

## 1.4 PHP als eigenständige Anwendung

Es ist wie bei anderen Interpretersprachen auch bei PHP möglich, den Interpreter standalone zu betreiben. Dies ermöglicht es, PHP-Skripte losgelöst von einem Webserver zu betreiben. Grundsätzlich sollte klar sein: PHP selbst erzeugt als Ausgabe lediglich Bytes, ob es sich dabei nun um reinen Text, HTML-Quelltext oder gar ein PDF-Dokument handelt, ist für PHP selbst erst einmal uninteressant – es macht lediglich das, was der Programmierer vorgibt.

PHP als alleinigen Interpreter zu verwenden, hat mehrere Makel und sollte wohlüberlegt sein. Zum Einen können natürlich alle Funktionen, die direkt auf eine HTTP-Verbindung abzielen, möglicherweise nicht benutzt werden. Dies trifft immer dann zu, wenn PHP wirklich eigenständig betrieben wird (und nicht etwa lediglich als CGI von Apache oder einem anderen Webserver aufgerufen wird). Selbstverständlich sind diese Funktionen aufrufbar, doch solange der Prozess, der aktuell PHP aufruft (also etwa die Kommandozeile), nicht damit umgehen kann, wird entsprechend die Ausgabe zum Teil unübersichtlich oder gar unnützlich. Und die Probleme können noch weitergehen: Schutzmechanismen, die etwa über den Webserver aktiviert werden können, stehen bei einem direkten Aufruf meist nicht zur Verfügung. Wird PHP mit einem Webserver betrieben, kann etwa bereits mit dem Modul `mod_rewrite` verhindert werden, dass bestimmte Clients auf PHP-Skripte zugreifen, und es kann mittels der durch den Server bereitgestellten Authentifikation der Benutzerkreis eingegrenzt werden (diese Authentifizierung ist in jedem Fall sicherer als eine selbst implementierte Login-Funktionalität).

Zum Anderen ist ein selbstständiger Betrieb von PHP aus sicherheitstechnischer Sicht durchaus etwas, was man, vor allem wenn man auf externe Tools oder Module, Klassen und Bibliotheken aus unsicheren Quellen angewiesen ist, in Betracht ziehen sollte, um einen eventuellen Schaden am System weiter zu begrenzen. Läuft diese Anwendung auf einem niedrig privilegierten Benutzer, so sind zumindest auf einem Linux-System die möglichen Schäden begrenzt (dies gilt auch für Windows-Systeme, doch wird dort leider von benutzerbasierten Zugriffsrechten selten Gebrauch gemacht). Stürzt PHP ab, wird es so auch wahrscheinlich keine anderen Prozesse in Mitleidenschaft ziehen, außer es wird beispielsweise eine rechenintensive Endlosschleife o.Ä. initiiert, die das System so stark auslastet, dass andere Prozesse nicht mehr zeitnah arbeiten können.

Und natürlich erweitert sich das Einsatzgebiet von PHP auch deutlich, wenn es ohne Webserver verwendet wird, da Skripte dann im Dauerbetrieb gestartet werden können und so beispielsweise eine Netzwerkserveranwendung realisierbar ist, die darauf wartet, dass sich Clients verbinden und Daten anfordern. Dieses Feature wird vor allem in Kapitel 12 beschrieben.

## 1.5 PHP mit alternativen Webservern

Mit anderen Webserversystemen (etwa dem Microsoft Internet Information Server oder dem Roxen Webserver) kann PHP entweder als CGI-Modul oder als SAPI-Modul betrieben werden.

SAPI steht dabei für Server Application Programming Interface, es handelt sich also um eine festdefinierte Schnittstelle, über die PHP-Daten vom Webserver bereitgestellt werden und über die PHP selbst auch wieder Daten an den Server zurückliefern kann.

Auch hier gilt wie bei der Symbiose zwischen Apache und PHP: SAPI ist eindeutig schneller als CGI, jedoch sollte die Konfiguration des Webservers genutzt werden, um gravierendere Angriffe auf das System zu vereiteln.

## 1.6 Sicherheitskonzepte für Webanwendungen

Schon immer gab es Versuche, in Datenverarbeitungsanlagen gewaltsam einzudringen und entweder an empfindliche Daten zu gelangen oder das System zu stören, so dass der Betreiber dieses mit hohem Aufwand wiederherstellen muss (und dabei eventuell auch Daten verliert).

Bei komplexen Systemen, die Webserver zweifellos sind, ist es allerdings nicht ausreichend, lediglich die Webanwendungen selbst sowie die Software des Webservers abzusichern. Einem Angreifer, der über einen anderen Weg in das System gelangt, würde es dann auf jeden Fall möglich sein, entsprechende Aktionen durchzuführen, die die Bereitstellung der Webanwendung beeinträchtigen.

Das Sichern der verwendeten PHP-Skripte sollte also nur ein Bestandteil eines Sicherheitskonzeptes sein. Hinzu kommt die restriktive Konfiguration des Webservers und des PHP-Moduls. Zudem sollte jeder Rechner, der Daten nach außen bereitstellt, durch eine Firewall abgesichert werden. Hier ist eine Hardware-Firewall vorzuziehen – doch eine softwarebasierte Version ist in jedem Fall immer noch besser, als ganz auf einen Schutz zu verzichten.

Es ist wichtig, eine Absicherung der Webanwendungen nicht als Aufgabe zu sehen, die schnell nebenbei erledigt werden kann. Wichtig ist hier die Anwendung eines Sicherheitskonzeptes; dies umfasst dann nicht nur die zu schützende Software bzw. die sensitiven Daten (egal, ob es sich nun um den Quelltext der Applikation

selbst oder um damit gewonnene kritische Daten, wie etwa personenbezogene Daten von Kunden, handelt), sondern das System, auf dem eine Webanwendung betrieben werden soll, als Ganzes. Je nach Umgebung und Wichtigkeit kann das »Ganze« somit sowohl lediglich für den einzelnen Server als auch für das gesamte Netzwerk stehen.

### Hinweis

Auf den folgenden Seiten wird das Thema IT-Sicherheit lediglich angerissen; mehr ist im Rahmen dieses Buches auch nicht möglich, denn schließlich liegt der Fokus auf den möglichen Sicherheitslücken in Webanwendungen und direkt betroffener Software und nicht in Grundsatztheorien über sichere Systeme. Falls Sie sich bisher nie mit diesem Thema beschäftigt haben, ist weitergehende Lektüre auf jeden Fall sinnvoll.

Jedes Sicherheitskonzept sollte die grundlegenden Ziele der IT-Sicherheit verfolgen. Haben Sie ein Konzept für Ihre Umgebung entwickelt (es gibt hierbei keine pauschale immer gleich effektive Lösung – Sicherheit ist individuell!), können Sie grundsätzlich feststellen, ob dieses Konzept allen Erfordernissen gerecht wird, indem Sie es bezüglich der folgenden Ziele prüfen; diese sollten stets erfüllt werden:

- **Datenschutz**
  - Vertraulichkeit: Daten dürfen nur von autorisierten Anwendern einsehbar sein.
  - Übertragungssicherheit: Lediglich der Anwender und das System sollen die Daten während der Übertragung auslesen. Die Einsicht online durch Dritte muss unterbunden werden.
  - Privatsphäre: Persönliche Daten und die Anonymität müssen gewährleistet sein. Wichtig ist hier auch das Recht auf informationelle Selbstbestimmung.
  - Datenschutzgesetze: Die Datenschutzgesetze müssen eingehalten werden.
- **Datensicherheit**
  - Deterministik: Hard- und Software sollten so funktionieren, wie es erwartet wird. Das Resultat sollte dabei immer dem gewünschten Ergebnis entsprechen.
  - Integrität: Software und Daten dürfen nicht unbemerkt verändert werden können.
  - Authentizität: Die Echtheit von Daten muss überprüfbar sein.
- **Organisatorische Sicherheit**
  - Verbindlichkeit: Es sollte für jeden Anwender offensichtlich sein, zu welchem Ergebnis eine bestimmte Aktion führt. Dabei sollte die gleiche Aktion stets zum selben Resultat führen (vgl. auch Deterministik).

- **Beweisfestigkeit:** Alle Übertragungen müssen so dokumentiert und gegebenenfalls vom Anwender bestätigt werden, dass sie auch im Falle eines Rechtsstreites Gültigkeit besitzen und ein Vorgang jeweils einwandfrei nachgewiesen werden kann.
- **Zugriffssteuerung:** Der Zugriff muss reglementiert sein; hierbei gibt es allerdings eine deutliche Abgrenzung zum Punkt »Vertraulichkeit«. Bei der Vertraulichkeit geht es lediglich darum, dass Daten nur von berechtigten Benutzern gelesen und/oder verändert werden können. Bei der Zugriffssteuerung geht es zudem darum, dass Benutzer, die diese Daten nicht verwenden können, auch keinen Zugriff darauf haben – also dass es etwa für Internetbenutzer nicht ersichtlich ist, dass diese Daten überhaupt existieren.
- **Verfügbarkeit:** Hier geht es primär um die Verhinderung von Datenverlust, sekundär sollte ein gutes Sicherheitskonzept natürlich auch nach Möglichkeit Systemausfälle verhindern.

Folgenden Risiken sollte zudem durch das Konzept explizit entgegnet werden:

- **Computerviren, Trojaner und Würmer:** Ein Anti-Viren-Programm ist für jedes System unabdingbar.
- **Backdoors:** Alle aus- und eingehenden Verbindungen, die nicht vom System und den verwendeten Anwendungen resultieren, sollten generell durch die Firewall gekappt werden.
- **Spionage, Hacking und Cracking:** natürlich ist es ein Ziel jedes Administrators, solche Angriffe zu verhindern. Hierfür kann es zwei entscheidende Schritte geben: Zum Einen sollte sämtliche Software, die auf dem Server verwendet wird (auch Zusatzmodule, die beispielsweise lediglich mit PHP verlinkt sind!), ständig aktualisiert werden. Das Intervall zur regelmäßigen Aktualisierung sollte drei Wochen betragen. Zum Anderen ist der Einsatz eines Intrusion Detection Systems empfehlenswert; dabei sollte sowohl das klassische IDS, das eingehende und ausgehende Netzwerkverbindungen analysiert, als auch eine Dateisystemüberwachung, die Konfigurations- und Programmdateien auf Veränderungen überprüft, eingesetzt werden.
- **Phishing und andere Identitätsangriffe:** Besonderes Augenmerk sollte – sofern mit einer Webanwendung personenbezogene Daten verarbeitet oder gespeichert werden – auf eindeutige Erkennbarkeit gelegt werden. Es ist erforderlich, dass Ihre Webanwendung für jeden Anwender erkennbar ist – dies kann z.B. durch ein SSL-Sicherheitszertifikat einer offiziellen Ausgabestelle sichergestellt werden.
- **Spoofing:** Verlassen Sie sich niemals auf Informationen, die Sie durch eine dritte Ebene erlangen können. Akzeptieren Sie also bei einer Authentifizierung beispielsweise niemals eine IP-Adresse als gegeben – jede Herkunftsinformation lässt sich im Zweifelsfall ohne großen Aufwand fälschen.

- **Höhere Gewalt:** Eine Absicherung gegen Stromausfall und Überspannung gehört auch zu einem Sicherheitskonzept, vor allem da durch Überspannung bedingt ein Datenverlust eintreten könnte.
- **Social Engineering:** Dieses Thema ist lediglich dann interessant, wenn es sich um eine Umgebung in einem größeren Unternehmen handelt oder der Support, der etwa Passwörter neu vergibt, ausgelagert wurde. Wenn ein Anwender eine Information oder gar eine Änderung an seinem Account anfordert, ohne dass er sich vorher über das System authentifiziert hat, sollte stets sichergestellt werden, dass es sich wirklich um diesen Benutzer handelt. Aber: Die übliche obligatorische Frage nach dem Namen des Haustiers oder dem Geburtsnamen der Mutter sollte hier keine Anwendung finden, da sich viele dieser Informationen über Suchmaschinen recherchieren lassen.

Die Erstellung und die anschließende – kompromisslose – Umsetzung des Sicherheitskonzepts sollten in Firmen selbstverständlich sein. Doch leider ist dies vor allem in mittelständigen Unternehmen nicht der Fall; da IT-Sicherheit als solches lediglich offensichtliche Kosten jedoch keinen sichtbaren Nutzen und schon gar keinen in der Bilanz monetär sichtbaren Erfolg bringt, wird auf dieses Thema entweder aus finanziellen oder zeitlichen Gründen gern verzichtet.

Doch auch der private Serverbetreiber sollte ein Konzept realisieren, auch wenn dies erst einmal mit sehr viel Aufwand verbunden ist. Dieser Aufwand scheint umso größer, wenn man betrachtet, dass er lediglich zur Sicherung eines Hobbys oder bestenfalls eines Nebenerwerbs dient. Doch die Gründe, die für mehr Sicherheit sprechen, sind bei einem privat betriebenen Server, der sensible Daten beinhaltet oder gar speichert, umso größer. Kommt es hier zu einem erfolgreichen Angriff und Daten werden entwendet und gar an zweifelhafte Dritte verkauft, können straf- und zivilrechtliche Folgen umso schwerwiegender sein.

Damit Sie einen Anhaltspunkt haben, wie Sicherheit in der Praxis gewährleistet werden kann, folgt nun eine Liste von Praktiken, die angewendet werden können, um ein Sicherheitskonzept umzusetzen; diese ist natürlich nicht vollständig und kann lediglich als Anregung dienen:

- **Software aktualisieren:** Wie bereits erwähnt, sollte jede benutzte Software in regelmäßigen Abständen aktualisiert werden.
- **Anti-Viren-Software verwenden:** Auch wenn lediglich Programme aus scheinbar zuverlässigen Quellen bezogen werden und niemand auf dem Server Anwendungen installieren oder ausführen kann, ist ein Virenprogramm ein Muss.
- **Diversifikation:** Es sollte möglichst viel Software von verschiedenen (am besten auch nicht marktführenden) Herstellern verwendet werden. Applikationen von Marktführern ist oft Ziel von Angriffen; das Risiko kann bereits etwas mini-

miert werden, wenn man sich softwaremäßig nicht vollkommen in die Hand eines Herstellers begibt.

- Firewalls verwenden: Auch schon mehrfach aufgeführt: Eine Firewall ist unabdingbar.
- Benutzerrechte einschränken: Vor allem unter Windowsbenutzern beliebt ist es, dass jeder Benutzer auf alles zugreifen darf. Für Server, die aus dem Internet zu erreichen sind, sollte ein differenziertes Rechtesystem verwendet werden. Vor allem ist es wichtig, dass die Benutzer, die die zentralen Dienste wie Webserver und Datenbankserver betreiben, entweder nur Leserechte auf die wirklich notwendigen Daten (dies gilt dann allen voran für den Webserver-User) oder nur volle Zugriffsrechte auf die zu ändernden Dateien (hiermit ist der Datenbankserver gemeint) erhalten.
- Sensible Daten verschlüsseln: Dies ist vor allem in Webanwendungen eher schwer zu realisieren. Werden Daten in einer Datenbank abgelegt, sollten eventuell im DBMS vorhandene Möglichkeiten der verschlüsselten Speicherung der Datenbankdateien genutzt werden. Werden Daten hingegen direkt in Dateien – etwa in Text- oder XML-Dateien – abgelegt, sollte auf jeden Fall in Betracht gezogen werden, diese zu verschlüsseln. Alternativ kann auch die gesamte Festplatte verschlüsselt werden. Hier muss jedoch beachtet werden, dass bei Problemen mit dem System an sich eventuell eine Entschlüsselung und somit ein Zugriff auf die Daten nicht mehr möglich ist.
- Sicherungskopien erstellen: Trotz aller Absicherungen von der unterbrechungsfreien Stromversorgung und Überspannungsschutz bis hin zum Booten des Betriebssystems von einer CD-ROM kann es dennoch zu einem Systemausfall kommen. Besonders wenn es zu Hardwaredefekten kommt und ein anderer Rechner verwendet werden muss, ist es notwendig, die bestehenden Daten relativ zeitnah und ohne Umstände (also z. B. den Umbau der Festplatte aus dem »alten« Server) bereitzustellen. Hierfür sollten regelmäßige Sicherheitskopien bereitstehen.
- Protokollierung: Um Fehlverhalten nachvollziehen zu können, sollten Protokolle – soweit sie sinnvoll sind und die Performance des Systems nicht zu stark beeinträchtigen – aktiviert werden. Allerdings ist es damit nicht getan; selbst wenn das System »reibunglos« funktioniert, sollen diese Protokolle regelmäßig durchgesehen werden, da sich vor allem Hardwaredefekte schleichend ankündigen und so bereits frühzeitig über Protokolldateien erkannt werden können.
- Überprüfung: IT-Sicherheit ist ein kontinuierlicher Prozess, der ständig angepasst werden muss. Es sollte also regelmäßig geprüft werden, ob die ergriffenen Maßnahmen noch zu den Risiken passen. Vor allem, wenn neue Software installiert wurde, sollte das Sicherheitskonzept gezielt geprüft werden.

- Sensibilisierung: Das beste System ist nutzlos, wenn alle Sicherheitsbestimmungen durch unvorsichtige Mitarbeiter leichtfertig umgangen werden. Auf dem Produktivsystem sollte niemals etwas getestet werden, es sollten auch keinesfalls Dateien mal eben für irgendjemand freigegeben werden – diese vorübergehenden Freigaben werden gern vergessen und ein böswilliger Dritter ist über die Bereitstellung dieser vielleicht sogar sensiblen Daten sehr dankbar.