Dr. Stefan Wohlfeil
Prof. Dr. Armin R. Mikler

Modul 32411
Sicherheit: Safety + Security

01867 Sicherheit im Internet II
21811 Fehlertoleranz in Computersystemen und Netzwerken

LESEPROBE

mathematik und informatik

FernUniversität in Hagen

Dr. Stefan Wohlfeil

Kurs 01867
Sicherheit im Internet II

LESEPROBE

mathematik und informatik

FernUniversität in Hagen

Inhaltsverzeichnis

Kapitel 1

Angriffe auf Rechner oder Netze

Der Autor: Prof. Dr. Stefan Wohlfeil, geb. 12.12.1964

- Studium der Informatik mit Nebenfach Elektrotechnik an der Universität Kaiserslautern (1984–1991)
- Wissenschaftlicher Mitarbeiter am Lehrgebiet Praktische Informatik VI der FernUniversität Hagen (1991–1998)
- Promotion zum Dr. rer. nat. (1997)
- Mitarbeiter in der Deutsche Bank AG, Abteilung TEC — The Advanced Technology Group (1998–2002)
- Professor in der Fakultät IV, Abteilung Informatik der Hochschule Hannover; Arbeitsgebiet: Sichere Informationssysteme (seit 2002)

1.1 Einführung

Liebe Fernstudentin, lieber Fernstudent,
herzlich willkommen beim zweiten Kurs über Sicherheit im Internet!

Diese Einführung soll Ihnen einen Überblick darüber geben, worum es im vorliegenden Kurs geht. Dieser Kurs gehört zum Bereich *M2 Computersysteme* und ist ein Teil des Moduls *Sicherheit – Safety & Security*.[1] Der vorliegende Kurs vertieft das Thema *Sicherheit*, in das bereits im Kurs *(01866) Sicherheit im Internet 1* eingeführt wurde.

Vorkenntnisse

Inhalt des Kurses und Vorkenntnisse: Dieser Kurs richtet sich an Informatik-Studierende und setzt die Kenntnis der Inhalte aus dem Kernbereich des Bachelor-Studiengangs Informatik voraus. Kenntnisse aus dem Kurs *(01866) Sicherheit im Internet 1* sind hilfreich. Konkret sollten Sie bereits wissen, wie ein Computer prinzipiell aufgebaut ist, was ein Betriebssystem typischerweise macht und welche Möglichkeiten sich durch die Vernetzung, wie beispielsweise im Internet, für Anwender bieten. Außerdem sollte Ihnen die Grundlagen des Internet und die grundsätzlichen Bedrohungen dort bereits bekannt sein.

[1] Stand dieser Information Mai 2014. Beachten Sie bitte immer auch die Hinweise zur Belegung und die Informationen aus dem Prüfungsamt.

Weiterhin sollten Sie Grundkenntnisse in der Softwareentwicklung haben. Dazu gehören Kenntnisse über die Entwicklungsprozesse sowie die Grundlagen der Programmierung in den Sprachen C, C++ und Java.

Inhalt

Die Kurseinheit 1 beschäftigt sich mit den möglichen Angriffen gegen Computer oder Netze. Hier werden Techniken vorgestellt, mit denen Angreifer versuchen Schaden anzurichten. Dazu gehören Techniken wie Abhören von Datenpaketen, Vortäuschen falscher Tatsachen, Einsatz von „Hacker"-Programmen oder „Knacken" von verschlüsselten Nachrichten.

Bezahlverfahren
digitale Münzen
Gesetze

Das Thema Benutzersicherheit wird in Kurseinheit 2 weiter vertieft. Dazu wird zunächst ein Überblick über wirtschaftliche Aspekte des Internets gegeben. Bei wirtschaftlichem Handeln geht es zumeist um Tausch, beispielsweise von Waren gegen Geld. Im Kurs werden verschiedene Bezahlverfahren vorgestellt, die im Internet eingesetzt werden können. Außerdem werden die technischen Grundlagen für die Realisierung von digitalem Geld vorgestellt. Anschließend werden gesetzliche und organisatorische Aspekte vorgestellt. Sie beginnt mit einem Überblick über relevante Gesetze und Verordnungen. Ihre Inhalte und die praktischen Auswirkungen werden, ohne in juristische Details zu gehen, vorgestellt.

VPN
IDS

In Kurseinheit 3 wird das Thema Anbietersicherheit vertieft. Die beiden vorgestellten Schwerpunkte sind (1) Virtual Private Networks (VPN) und (2) Intrusion Detection Systeme (IDS). Mit Hilfe von VPNs kann man Rechner an weit auseinanderliegenden Orten sicher über ein potentiell unsicheres Netz (z. B. das Internet) miteinander verbinden.

Mit Hilfe von Intrusion Detection Systemen sollen Administratoren Angriffe auf die eigenen Rechner, bzw. das eigene Netz erkennen. Die Funktionsprinzipien und die Konfiguration solcher Systeme werden in dieser Kurseinheit vorgestellt.

Erstellung sicherer Systeme

Am Ende der Kurses werden in Kurseinheit 4 Aspekte der Entwicklung sicherer Systeme behandelt. Für einen Informatiker sind neben der Technologiekenntnis auch die Vorgehensmodelle bei der Erstellung sicherer Systeme von Bedeutung. Es werden Prozesse vorgestellt, die beim Bau sicherer Systeme hilfreich und nützlich sind. Einige der typischen Programmierfehler, die dann zu Sicherheitsproblemen führen werden vorgestellt. Dazu kommen dann Hinweise, wie man diese Fehler vermeiden kann. Grundlagen für ein sicheres System sind Bedrohungsanalysen, deren Ergebnisse in die Systemarchitektur einfließen. Zusammen mit einer sicheren Implementierung führt das zu sichereren Systemen. Den Abschluss des Kurses bilden Hinweise auf den sicheren Betrieb.

Ergänzende Materialien: Das Thema Sicherheit im Internet ist derart umfangreich, dass es auch in diesem vertiefenden Kurs nur in Ausschnitten behandelt werden kann. Ziel des Kurses ist es, dass Sie die Grundlagen des Themengebietes kennenlernen und Sie sich dann darauf aufbauend tiefer in die Materie einarbeiten können. Dazu gibt es verschiedene weitere Informationsquellen.

Bücher

Die Menge an Büchern zum Thema Security wächst sehr schnell. Ausgehend vom Literaturverzeichnis dieses Kurses sollten Sie in der Universitätsbibliothek das eine oder andere Buch ausleihen und durchschauen. Aktuellste Bücher kann man bei den verschiedenen Buchhändlern im Internet suchen. Dort findet man u. U. auch Rezensionen der Bücher vor.

Überhaupt ist das Internet eine nahezu unerschöpfliche Quelle an Informationen zum Thema Security. Im Kurs werden eine Reihe von Verweisen auf interessante Seiten im Internet genannt. Wenn Sie Zugang zum Internet haben, nehmen Sie sich doch die Zeit und schauen sich die eine oder andere Seite an. Ich hoffe, dass die Verweise noch stimmen, wenn Sie den Kurs lesen. Das Internet ändert sich ständig, so dass es gut sein kann, dass Sie einmal eine Seite nicht finden. In diesem Fall sollten Sie eine der vielen Suchmaschinen wie z. B. *bing* oder *google* konsultieren. Vielleicht hat sich ja nur die Adresse der Seite leicht verändert. Informieren Sie dann auch bitte die Kursbetreuer, damit der Kurstext aktualisiert werden kann. Die Namen und Sprechstunden der Kursbetreuer wurden Ihnen im Anschreiben zusammen mit dieser Kurseinheit genannt.

Internet

1.2 Sniffing

Systeme, die Datenverkehr auf einem Netz mitlesen, nennt man **sniffer** von schnüffeln (engl. **to sniff**). Die Möglichkeiten um Datenverkehr zu belauschen hängen stark von der verwendeten Übertragungstechnologie ab. In Funknetzen (WLAN) ist es einfacher, die Signale zu empfangen als in drahtgebundenen Netzen. Im zweiten Fall braucht man zumindest einen physischen Zugang zu einer Leitung des Netzes.

sniffer

Mit einem **hub** realisierte lokale Netze lassen sich vergleichsweise einfach belauschen. Ein hub verteilt alle ankommenden Datenpakete an alle angeschlossenen Geräte. Die Ethernet-Karten in PCs sind normalerweise so konfiguriert, dass sie Pakete, die nicht für diesen PC gedacht sind, verwerfen. Man kann eine Ethernet-Karte jedoch auch so konfigurieren, dass sie alle Pakete entgegen nimmt. Jedes Paket wird dann an die nächsthöhere Schicht im Protokoll-Stack geleitet. Dieser Betriebsmodus wird **promiscuous mode** genannt. Normalerweise benötigt man Administratorrechte, um diesen Modus einzuschalten.

hub

promiscuous mode

Die so gelesenen Datenpakete werden von einem sniffer-Programm gesammelt und analysiert. Da die Analyse mit einigem Aufwand verbunden ist, werden die Pakete häufig erst einmal gesammelt, d. h. in eine lokale Datei geschrieben, und später analysiert. Dabei versucht der sniffer bekannte Protokolle, wie beispielsweise telnet, ftp oder http, in den gespeicherten Paketen zu entdecken. Ein Beispiel für die Ausgabe des Programms *Wireshark* zeigt Abbildung 1.1.

Im unteren Bereich des *Wireshark*-Fenster in Abbildung 1.1 sieht man die Bytes des Datenpakets, einmal in hexadezimaler Darstellung und einmal als ASCII-Zeichen. Beides ist invers, also in weißen Buchstaben dargestellt. Der Anfang des angezeigten Pakets enthält die Header-Informationen. Sie können normalerweise nicht sinnvoll als ASCII-Zeichen dargestellt werden. Statt dessen zeigt das Programm einen Punkt an. Weiter hinten im Datenpaket stehen dann die Informationen aus dem Rumpf. Sie sind häufig als ASCII-Zeichen kodiert und können somit auch vom Menschen gelesen und interpretiert werden. In Abbildung 1.1 enthält der Rumpf des Pakets den bei der ftp-Anmeldung eingegebenen Benutzernamen (wohlfeil) im Klartext.

Das Ergebnis der genauen Analyse des Pakets ist in der Mitte von Abbildung 1.1 dargestellt. Man sieht beispielsweise:

- Die MAC-Adressen der Netzwerkkarten in der Zeile *Ethernet II*

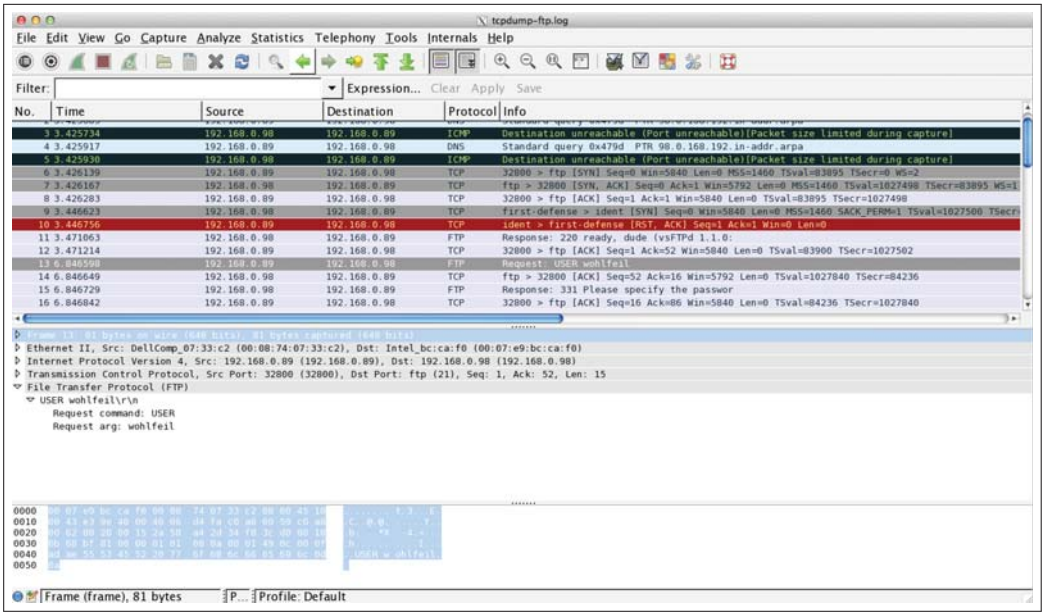

Abbildung 1.1: Ausgabe des Programms *Wireshark*

- die Quell-IP-Adresse (Src) und die Ziel-IP-Adresse (Dst) in der Zeile *Internet Protocol Version 4*
- die Portnummern (Source Port, Destination Port) in der Zeile *Transmission Control Protocol*
- weitere Verwaltungsdaten (Sequence number, Acknowledgement number, etc.) aus derselben Zeile wie die Portnummern

Selektiert man eine Zeile im mittleren Bereich mit der Maus, so werden die zugehörigen Bytes im unteren Bereich invers dargestellt. Da das ganze Paket selektiert ist, sind alle Bytes invertiert dargestellt.

Der obere Bereich von Abbildung 1.1 zeigt in einer Zeile zusammengefasst die wichtigsten Informationen der einzelnen Pakete. Man erkennt in Zeile No. 6, dass der Rechner mit der IP-Adresse 192.168.0.89 eine TCP-Verbindung zum Rechner mit der IP-Adresse 192.168.0.98 aufbauen will. Dabei findet der typische Handshake in drei Phasen (engl. **three way handshake**) zum Verbindungsaufbau statt. Er besteht aus (siehe Zeilen 6–8):

1. Einem syn-Paket vom Absender an den Zielrechner
2. Einem syn/ack-Paket vom Zielrechner an den Absender
3. Einem ack-Paket vom Absender an den Zielrechner

Danach wird die Begrüßungsmeldung vom Zielrechner, in diesem Fall einem ftp-Server, übertragen. In Zeile 13 sieht man, dass die Anmeldung am ftp-Server unter der Benutzer-Kennung `wohlfeil` erfolgt. Zeile 17 (absichtlich nicht in Abbildung 1.1 mit aufgenommen :-) enthält das Paket mit dem Passwort des Benutzers im Klartext.

Das Passwort und die Benutzer-Kennung sind für einen Angreifer wahrscheinlich am wertvollsten. Mit ihnen kann er sich später auch am ftp-Server anmelden und vorgeben, der Benutzer `wohlfeil` zu sein. Allerdings findet man auch in den mitgelesenen Datenpaketen bereits viele weitere wichtige

Informationen. Neben den Verzeichnisnamen und Dateinamen werden auch die angeforderten Dateien im Klartext übertragen. Sie können vertrauliche Informationen enthalten.

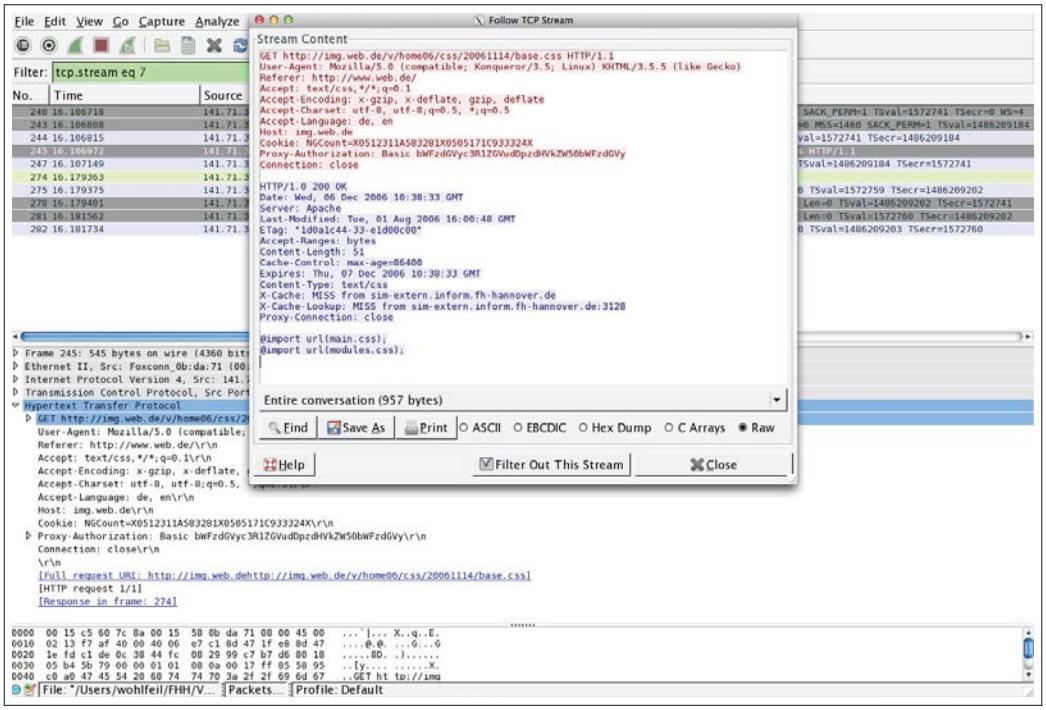

Abbildung 1.2: Zusammenfassung eines TCP-Datenstroms

Mit *Wireshark* kann man auch alle Pakete, die zu *einer* TCP-Verbindung gehören, zusammen anzeigen lassen. Abbildung 1.2 zeigt eine Zusammenfassung der Nutzlast aller Pakete einer TCP-Verbindung. Die rot hinterlegten Zeilen sind ein HTTP-Request, der von einem Browser an einen Squid-Proxy-Server gerichtet ist. An der vorletzten roten Zeile

```
Proxy-Authorization: Basic bWFzdGVyc3R1ZGVudDpzdHVkZW50bWFzdGVy
```

erkennt man, dass der Proxy eine Benutzerauthentisierung verlangt hat. Die vom Benutzer im Browser eingegebene Benutzerkennung sowie das Passwort hat der Browser konkateniert und dann einfach kodiert. Das Kodierungsverfahren basiert auf BASE64 und lässt sich einfach invertieren. Ein Angreifer erfährt somit aus dem String hinter `Basic` eine gültige Kombination von Benutzerkennung und Passwort für den Proxy.

Wireshark funktioniert natürlich nur, wenn der Zugriff auf ein gemeinsames Übertragungsmedium gegeben ist. Ein solches Medium kann ein klassischer hub oder ein drahtloses lokales Netz (engl. **wireless LAN**) sein. Nur dann kann der Angreifer alle Datenpakete, also auch die, die nicht an seine Adresse gerichtet sind, mitlesen. Durch den Einsatz eines **Switches** kann man diese Art von Angriffen leider nicht völlig vereiteln, aber doch wenigstens erschweren.

Switches

Ein Switch verteilt eingehende Pakete normalerweise nicht an alle angeschlossenen Geräte, sondern nur an den im Paketkopf genannten Empfänger. Dazu muss der Switch natürlich wissen, welche Geräte mit welchen Adressen an seinen Anschlüssen eingesteckt sind. Dies lernt der Switch während des Betriebs, indem die Absenderadressen von eingehenden Paketen im Switch gespeichert werden. An dieser Stelle können nun weitere Angriffe einsetzen, siehe dazu Abschnitt 1.3.1.

1.3 Spoofing

Maskierungsangriff

Unter dem Begriff **Maskierungsangriff** (engl. **spoofing**) versteht man das Vortäuschen einer falschen Identität. Dabei kann der Angreifer eine falsche IP-Adresse seines Systems oder einen falschen DNS-Namen seines Systems vortäuschen. In den folgenden Abschnitten werden beide Verfahren kurz vorgestellt.

1.3.1 IP spoofing

DHCP

Jeder Administrator eines Rechners kann eine falsche IP-Adresse angeben. Die IP-Adresse kann automatisch, z. B. von einem **DHCP**-Server (engl. **Dynamic Host Configuration Protocol**) vergeben werden oder sie wird vom Administrator im System konfiguriert. Somit kann ein Administrator jederzeit problemlos eine IP-Adresse aus dem aktuellen Subnetz vergeben.

ARP-Pakete gehen an alle angeschlossenen Geräte. Das hat zur Folge, dass alle Geräte nach dem ersten Broadcast die MAC-Adresse des Absenders kennen. Ein Angreifer kann nun gefälschte ARP-Pakete verschicken. Das kann eine gefälschte Antwort auf eine ARP-Anfrage oder einfach eine gefälschte ARP-Nachricht sein. Besonders subtil wird dieser Angriff, wenn der Angreifer vorgibt,

Router

der **Router** zu sein. An den Router werden alle Pakete geschickt, deren Ziel nicht im lokalen Netz liegt. Der Angreifer muss allerdings darauf achten, die abgefangenen Pakete anschließend an den richtigen Empfänger weiter zu leiten. Andernfalls würden Pakete verloren gehen und der Absender könnte den Angriff bemerken.

1.3.2 DNS spoofing

Rechner im Netz werden normalerweise nicht über ihre MAC-Adresse direkt angesprochen, sondern über IP-Adressen. Numerische IP-Adressen können sich Menschen aber schlechter merken als Namen (Zu welchem Rechner gehört beispielsweise die Adresse 195.71.11.67?[2]). In der Regel werden daher symbolische Namen benutzt. Für das Internet ist ein hierarchischer Namensraum definiert.

Man kann sich diesen Namensraum als Baum vorstellen, an dessen Wurzel

domain

eine namenlose Wurzel-Domäne steht. Der Begriff **domain** bezeichnet immer einen Teilbaum im hierarchischen Namensraum. Unter der Wurzel gibt es dann

top level domains

die sogenannten **top level domains** wie beispielsweise `de`, oder `com`, usw. Unter diesen Domains gibt es dann Unterdomänen, usw. Einen kompletten DNS-Namen liest man von unten nach oben und trennt die Domain-Namen durch einen Punkt.

Domain Name Service

forward zone

reverse zone

Die Zuordnung von Namen zu IP-Adressen wird vom **Domain Name Service** (DNS) vorgenommen. Ein DNS-Server verwaltet zwei Datenbanken/Tabellen. Die Abbildung von Name auf IP-Adresse wird in der **forward zone** Tabelle gespeichert. Die umgekehrte Abbildung von IP-Adresse auf Name wird in der **reverse zone** Tabelle verwaltet. Das Design von DNS verlangt keine Maßnahmen zur Überprüfung der Konsistenz dieser Tabellen. Weiterhin findet in der Regel keine sichere Authentisierung bei der Erstellung oder dem Austausch der Informationen in den Tabellen statt.

Probleme

Gefälschte oder inkonsistente Einträge können zu folgenden Problemen

[2] Am 16.8.2010 gehörte diese Adresse zum Namen `www.spiegel.de`

führen:

1. Unter UNIX kann ein Benutzer/Administrator anderen Rechnern trauen und einen Verbindungsaufbau von diesen Rechnern *ohne* explizite Authentisierung erlauben. Die Kommandos `rlogin` oder `rsh` haben diesen Mechanismus implementiert. Die Spezifikation der vertrauenswürdigen Rechner geschieht über DNS-Namen.

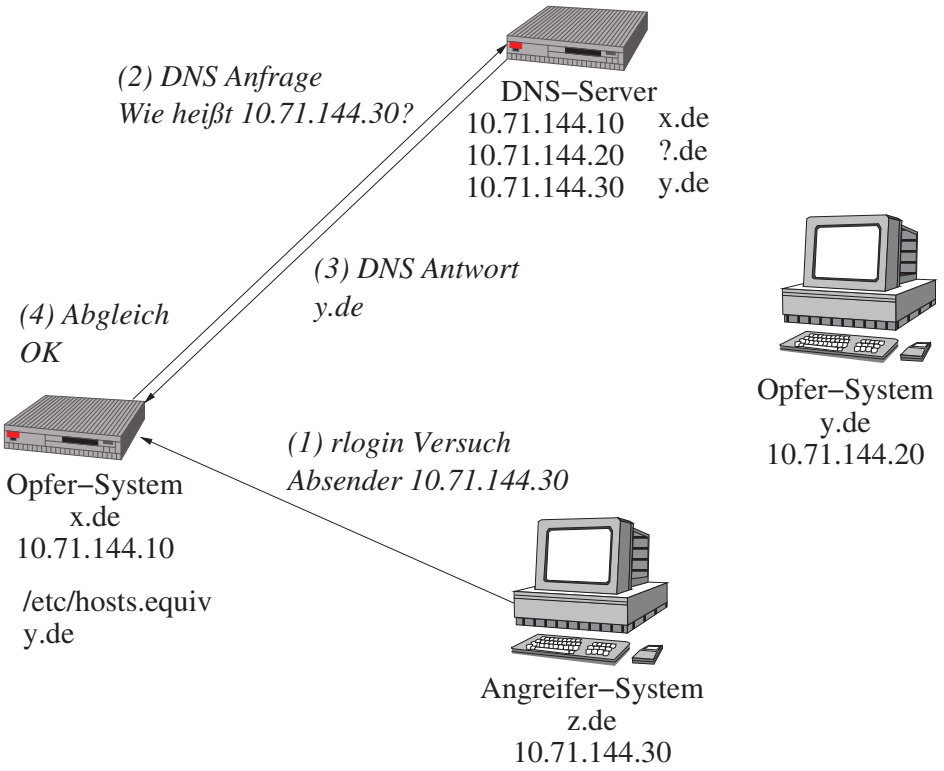

Abbildung 1.3: Angriff auf reverse zone

Das folgende Beispiel (siehe Abbildung 1.3) zeigt, welche Bedingungen erfüllt sein müssen und was beim Verbindungsaufbau genau passiert. Auf dem Rechner `x.de` soll dem Rechner `y.de` vertraut werden. Dazu wird auf dem Rechner `x` in der Datei `/etc/hosts.equiv` der Name `y.de` eingetragen.

Der Angreifer `z.de` soll nun versuchen, eine Verbindung mit `x.de` aufzunehmen und dabei vorgeben, er sei Rechner `y.de`. Dazu muss der Angreifer die reverse zone Tabelle des DNS-Servers verändern. Wenn `z.de` das `rlogin` Kommando ausführt, dann steht in den IP-Paketen an `x.de` die echte IP-Adresse von `z.de` als Absender. `x.de` fragt den DNS-Server nach dem Namen zur Absender IP-Adresse. Liefert der DNS-Server nun (fälschlicherweise) den Namen `y.de` zurück, so prüft `x.de` nur noch, ob dieser Name in `/etc/hosts.equiv` steht. Falls ja, so wird die Verbindung ohne Authentisierung aufgebaut.

Bei diesem Angriff gibt sich der Angreifer als falscher Absender aus.

2. Schafft es ein Angreifer, die forward Tabelle des DNS-Servers mit falschen Einträgen zu füllen, so kann er fälschlicherweise das Ziel von Verbindungsaufbauwünschen werden. So könnte der Angreifer eine komplette Website

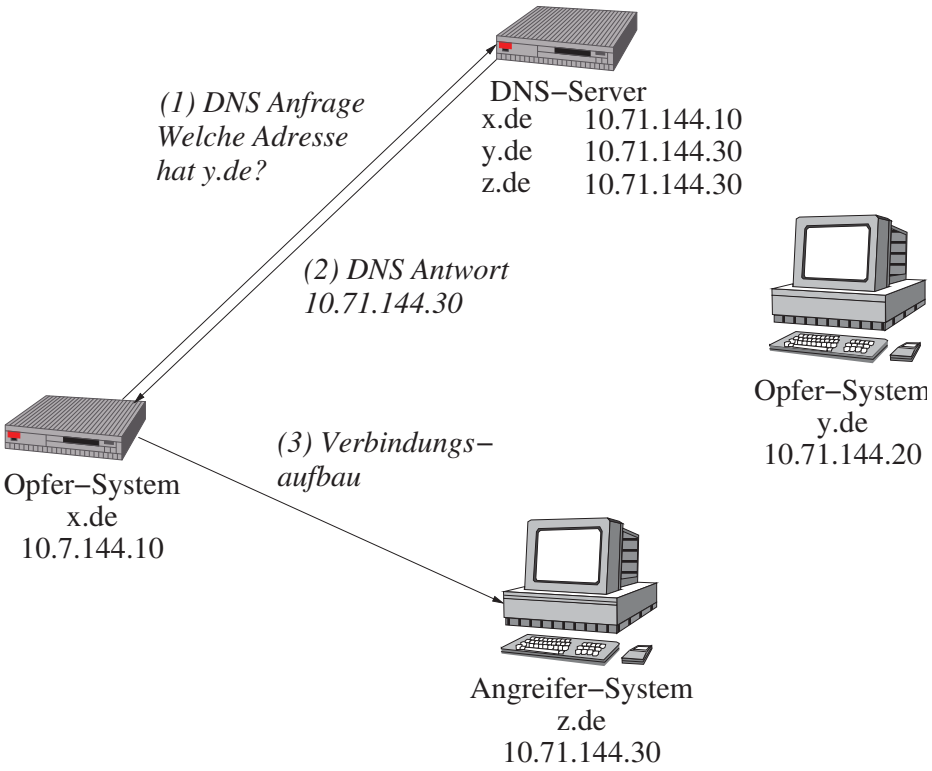

Abbildung 1.4: Angriff auf forward zone

auf dem eigenen Rechner nachbauen, beispielsweise einer E-Commerce Firma, einer Behörde oder einer Bank.

Abbildung 1.4 zeigt den Ablauf dieses Angriffs. Der Angreifer `z.de` hat die forward zone Tabelle des DNS-Servers so manipuliert, dass seine IP-Adresse unter dem Namen des Opfers (`y.de`) eingetragen ist. Der Rechner `x.de` möchte nun eine Verbindung zu `y.de` aufbauen und erfragt beim DNS-Server die IP-Adresse von `y.de`. Der DNS-Server liefert die Adresse von `z.de` und `x.de` baut eine falsche Verbindung auf.

Dieser Angriff kann drei Schaden-Typen zur Folge haben: (1) *Reputationsschaden:* Der Angegriffene könnte durch willkürliche Änderungen an den Inhalten verunglimpft werden. (2) *Umsatzschaden:* Durch Umleiten von Datenverkehr kann im E-Commerce Umsatz (und damit auch Ertrag) verhindert werden. Leitet ein Buchhändler beispielsweise den Verkehr von `amazon.de` auf seine Rechner, so erschleicht er sich dadurch möglicherweise neue Geschäfte. (3) *Vertraulichkeitsschaden:* Ist die Website interaktiv, d.h. es erfolgen Benutzereingaben, so kann der Angreifer an vertrauliche Informationen gelangen. Beim Internet-Banking gibt der Kunde seine
PIN **PIN** oder **TAN** ein und geht davon aus, dass diese Informationen nur
TAN an die Bank übertragen werden und nicht an einen Dritten.

Für die korrekte „Funktion des Internets“ ist ein korrekt arbeitendes DNS eine unverzichtbare Voraussetzung. Internet Service Provider (ISPs) oder Staaten können durch Manipulationen des DNS die Internetnutzung für ihre Kunden, bzw. Einwohner verhindern. Möchte man den Zugriff auf unliebsame Webseiten verhindern, so ändert man einfach im DNS die IP-Adresse zum unliebsamen Namen auf einen anderen Wert. Surfer, die `http://www.unliebsam.xy` eingeben werden dann nicht mit dem unliebsamen Rechner verbunden, sondern

bekommen eine andere Seite angezeigt. Natürlich können Benutzer immer noch durch die Angabe der IP-Adresse des Rechners in der URL die Seite abrufen. Dazu muss man die IP-Adresse erst einmal kennen. Außerdem bekommt man ein Problem, wenn man weitere Seiten durch Anklicken von URLs aufrufen will. Die „links“ in HTML-Seiten enthalten i. d. R. die DNS-Namen der Rechner und nicht die IP-Adressen.

Da DNS-Anfragen und DNS-Antworten weder verschlüsselt noch signiert sind, ist ihre Authentizität und ihre Integrität nicht gesichert. Ein Angreifer muss keinen DNS-Server manipulieren, sondern er kann auch die Nachrichten manipulieren. Um diese Art von Angriff zu verhindern, wurde eine Erweiterung des DNS-Protokolls definiert. Sie heisst **DNSSEC**. Die zugrundeliegende Idee besteht darin, dass DNS-Daten (konkret die Informationen zu einer DNS zone) digital signiert werden und jeder Empfänger durch Überprüfen der Unterschriften prüfen kann, ob die Daten authentisch und integer sind.

DNSSEC

Die Herausforderung bei der Einführung von DNSSEC liegt darin, dass DNS ein verteiltes System ist. Alle DNS-Server müssten es benutzen, d. h. die Server-Software muss aktualisiert werden und Schlüsselpaare für die Signierung müssen erstellt werden. Wie in public-key-Infrastrukturen (PKI, vergleiche Kurs *(01866) Sicherheit im Internet 1*) üblich muss man dann auch noch sicherstellen, dass man den korrekten öffentlichen Schlüssel der Gegenseite besitzt. Nur dann kann man die digitalen Signaturen prüfen.

In DNSSEC sind zwei Schlüsseltypen vorgesehen:

1. **key signing keys**. Diese Schlüssel werden nur dazu benutzt, andere Schlüssel digital zu signieren. Signiert beispielsweise die Wurzeldomäne (engl. **root domain**) mit ihrem privaten Schlüssel den Schlüssel einer der obersten DNS-Domäne (engl. **top level domain**), wie z. B. die `.de`-Domäne, dann bedeutet das, dass die Wurzeldomäne (engl. **top level domain**) den Verantwortlichen für die .de-Domäne vertraut und die .de-Domänendaten mit dem entsprechenden Schlüssel signiert sind.

   key signing keys

2. **zone signing keys**. Mit diesen Schlüsseln werden die eigentlichen DNS-Daten signiert. Eine zone bezeichnet hier einen Ausschnitt aus dem DNS-Namensraum, nämlich den Ausschnitt, für den ein DNS-Server zuständig ist. Dieser DNS-Server liefert also die „gültigen“ (engl. **authoritative**) Antworten zu Anfragen zu dieser Zone. Der Teilbaum `inform.fh-hannover.de` ist beispielsweise eine Zone und der DNS-Server der Abteilung Informatik ist für diese Zone zuständig. Der Administrator richtet neue Rechner ein, weist IP-Adressen und DNS-Namen zu und trägt diese Informationen in den DNS-Server ein.

   zone signing keys

Damit eine Antwort nun überprüft werden kann, braucht der Prüfende den öffentlichen Schlüssel des Unterzeichners. Einen solchen öffentlichen Schlüssel gibt es für *jede* DNS-Zone. Damit man nun nicht alle diese öffentlichen Schlüssel kennen muss, kann man auch eine Vertrauenskette (engl. **chain of trust**) benutzen. Kennt man den öffentlichen key-signing-Schlüssel einer Domäne (z. B. `fh-hannover.de`), so kann man den DNS-Daten einer Unter-Domäne (z. B. `inform.fh-hannover.de`) vertrauen, sofern sie mit einem Schlüssel signiert sind, der vom o. g. key-signing-Schlüssel signiert ist.

Im Juli 2010 wurden die Root-Server des DNS auf DNSSEC umgestellt. Als nächstes müssen die Betreiber der Top-Level-Domänen ihre Systeme auf

DNSSEC umstellen, dann die der Unterdomänen, usw. Am Ende müssen die DNS-Benutzer dann „nur noch“ die key-signing-Schlüssel der Wurzeldomäne kennen und können dann die DNS-Daten überprüfen.

Weitere Informationen zu DNSSEC finden Sie bei Kolkman [Kol09] oder in den RFCs zu DNSSEC. Als Startpunkt dient dort RFC 4035 *Protocol Modifications for the DNS Security Extensions*. Aktuell (Stand Mai 2014) gibt es über 25 RFCs in deren Titel (oder bei den Schlüsselwörtern) das Wort DNSSEC vorkommt.

1.4 Wörterbuchangriffe

Bei diesem Angriffstyp versucht der Angreifer, das Passwort eines legitimen Benutzers durch systematisches Ausprobieren herauszufinden. Dabei gibt es zwei verschiedene Strategien:

Strategien

1. Ausprobieren von bekannten Wörtern und ihren Abwandlungen, bzw. Kombinationen.

2. Ausprobieren aller möglichen Zeichenketten.

Die erste Methode setzt darauf, dass Benutzer ein Passwort wählen das sie sich einfach merken können. Oft sind das Wörter, die auch in Wörterbüchern wie dem *Duden* vorkommen. Im Internet existieren Wörterbücher in elektronischem Format für verschiedene Sprachen (Deutsch, Englisch, etc.) oder zu verschiedenen Themengebieten (Biologie, Technik, etc.). Programme können diese Wörter systematisch ausprobieren. Auch können dabei Variationen wie veränderte Groß-/Kleinschreibung oder Ersetzungen (z. B. Buchstabe O durch Ziffer 0) mit geprüft werden.

Die zweite Methode zählt alle möglichen Zeichenketten auf und prüft sie. Dabei werden natürlich auch alle bekannten Wörter vorkommen, jedoch besteht die Mehrzahl überwiegend aus „unsinnigen“ Zeichenketten. Bei einer Größe des Alphabets von x und einer Wortlänge von n gibt es x^n verschiedene Wörter. Für $x = 64$ Zeichen und eine Länge von 8 sind dies bereits 64^8. Dauert jeder Test nun 1 Millisekunde, so braucht man ca. $64^8/1000 \approx 281$ Milliarden Sekunden; das entspricht etwa 3.25 Millionen Tagen. Bei ausreichend langen Passwörtern ist diese Angriffsart nicht mehr praktikabel.

brute force

Aufwand

Bei beiden Methoden wird eine Hash-Funktion auf das zu prüfende Wort angewendet (siehe z. B. Kurs *(01866) Sicherheit im Internet 1* oder Kurs *(01801) Betriebssysteme und Rechnernetze*). Die Hash-Funktion ist dieselbe, die auch das Betriebssystem auf ein Passwort anwendet, bevor es das Passwort speichert. Der Hash-Wert jedes zu prüfenden Wortes wird mit jedem gespeicherten Hash-Wert verglichen. Dazu ist natürlich der Zugriff auf die gespeicherten Hash-Werte erforderlich. Unter Windows NT und seinen Nachfolgern gibt es drei Möglichkeiten, um an die Hash-Werte der Passwörter zu kommen:

1. Auszug aus der Windows Registry erstellen:

   Mit Hilfe spezieller Hilfsprogramme (z. B. *pwdump*) kann man die Hash-Werte aus der Systemdatenbank (Registry) oder dem zentralen Verzeichnis (genannt Active Directory) lesen. Dafür sind allerdings Administrator-Rechte sowie evtl. direkter, physischer Zugang zu dem Rechner erforderlich.

2. Auszug aus der SAM-Datei erstellen:

   Windows speichert die Benutzerdaten einschließlich Hash-Wert des Passworts in der SAM-Datei. Der Zugriff auf diese Datei ist bei laufendem Windows allerdings nicht möglich. Das Betriebssystem hat diese Datei geöffnet und exklusiv für sich reserviert. Somit kann man die Datei weder lesen noch kopieren.

   Allerdings wird diese Datei mit gesichert. Man kann also eine Sicherung erstellen und daraus dann die Datei kopieren. Die Sicherung wird über das Betriebssystem selbst erstellt, so dass die oben genannte exklusive Reservierung ohne Bedeutung ist.

   Alternativ kann man auf dem Rechner auch ein Linux-Live-System von DVD starten, die Windows-Partitionen der Festplatte unter Linux einbinden (engl. **to mount**) und dann die Datei kopieren. Um das zu erschweren, kann Windows diese Datei auch zusätzlich noch symmetrisch verschlüsselt auf der Festplatte speichern.

3. Abhören des Netzverkehrs:

   Bei der Anmeldung an einem Windows Client PC gibt der Benutzer sein Passwort ein. Die Prüfung des Passworts wird vom **Domain Controller** durchgeführt. Der Domain Controller ist eine Server-Maschine, auf der unter anderem auch die Benutzerdaten verwaltet werden. Die Kommunikation zwischen Windows Client und Domain Controller ist im **Server Message Block (SMB)** Protokoll geregelt. Dabei kann das Passwort verschlüsselt oder unverschlüsselt übertragen werden. Das Programm *Cain & Abel* kann beispielsweise den Netzverkehr mitschneiden und gleichzeitig versuchen, die Passwörter zu knacken.

   Domain Controller

   Server Message Block (SMB)

Unter UNIX stehen die Benutzer-Kennung und die Hash-Werte der Passwörter in der Datei `/etc/passwd`. Diese Datei darf jeder lesen. Um einen Wörterbuchangriff auf die Passwörter zu verhindern, werden die Hash-Werte der Passwörter heute in einer getrennten Datei, der sogenannten Shadow-Datei gespeichert. Sie kann nicht mehr von jedem gelesen werden.

Mit dem frei verfügbaren Programm *john* kann man den Inhalt einer Datei mit möglichen Passwörtern (Wortliste) auf Übereinstimmungen mit den Hash-Werten in der SAM-Datei prüfen. Daher ist es besonders wichtig, dass Sie ein gutes Passwort wählen (siehe Kurs *(01866) Sicherheit im Internet 1*), das insb. in keinem Wörterbuch steht.

1.5 Buffer Overflow Angriffe

Ein Speicherüberlauf (engl. **buffer overflow**) ist ein häufiger Grund für Systemabstürze. Dabei ist in einem Programm ein Speicherbereich fester Größe definiert, in den dann Daten *ohne* Längenprüfung geschrieben werden. Sind diese Daten länger als der Speicherbereich, so werden andere Teile des Speichers überschrieben. Dadurch können verschiedene Fehler auftreten.

Fehler

Programmabsturz: Falls der überschriebene Bereich binären Programmcode enthielt, der nun durch andere Zeichen überschrieben wurde, so ist die Wahrscheinlichkeit groß, dass diese Zeichen kein gültiger Maschinencode

für die CPU sind. In diesem Fall stürzt das Programm einfach ab, sobald der Code im überschriebenen Bereich ausgeführt wird.

Ausführen anderen Programmcodes: Falls der Angreifer eigenen gültigen Maschinencode an die Stelle des bisherigen Codes setzt, dann wird dieser neue Maschinencode ausgeführt. In diesem Fall tut das Programm etwas anderes als es normalerweise machen würde.

Modifikation von Daten: An der Stelle, an der der Speicherüberlauf auftritt, können auch Daten stehen, die verändert werden und dann zu inkonsistentem Verhalten des Programms führen oder falsche Ausgaben des Programms erzeugen.

Beispiel Ein Beispiel für den dritten Fall (aus [Bie06]) ist:

```
/* Hier das Passwort aus der Datenbank lesen */
char    origPasswort[12] = "Geheim\0";
char    userPasswort[12];

/* liest Benutzereingabe vom Terminal */
gets(userPasswort);

if (strncmp(origPasswort, userPasswort, 12) != 0)
{
        printf("Falsches Passwort!\n");
        exit(-1); /* Programm beenden */
}
/* Benutzer authentisiert */
```

Durch die Eingabe von mehr als 12 Zeichen kann der Benutzer in diesem Beispiel die Variable `origPasswort` überschreiben. Das gilt unter der Annahme, dass die Adresse von userPasswort[12+i] mit der Adresse origPasswort[i] übereinstimmt. Sie ist bei der Stack-Belegung der meisten Computer erfüllt. Falls nicht, dann existiert die Schwachstelle wenn die Reihenfolge der Variablendefinitionen vertauscht wird. Gibt der Benutzer nun genau 24 Zeichen ein, bei denen die vordere und die hintere Hälfte identisch sind, also beispielsweise „oberprogrammoberprogramm“, dann wird der Vergleich in der IF-Anweisung positiv ausfallen. Der Angreifer würde authentisiert und das Programm verhält sich anders als es sollte.

Ursache Der Grund hierfür ist, dass prozedurale Programmiersprachen einen Laufzeitstapel (engl. **runtime stack**) (siehe Abbildung 1.5) benutzen. Der Hauptspeicher wird linear adressiert, d. h. jede Speicherzelle bekommt eine Adresse zwischen 0 und *Speichergröße*. Der Platz für statische Daten wird an einem Ende des Adressraumes freigehalten, so dass ein zusammenhängender Adressbereich für die dynamischen Daten übrig bleibt. Häufig gibt es zwei „Typen“ von dynamischen Daten:

1. Vom Programm zur Laufzeit *explizit* angeforderte Speicherbereiche: In prozeduralen Sprachen wie Modula 2 oder C existieren Systemfunktionen wie `ALLOCATE` oder `malloc`, mit denen Speicherbereiche angefordert werden können. In Objekt-orientierten Sprachen wie C++ oder Java wird neuer Speicher für Objekte durch den Operator `new` angefordert.

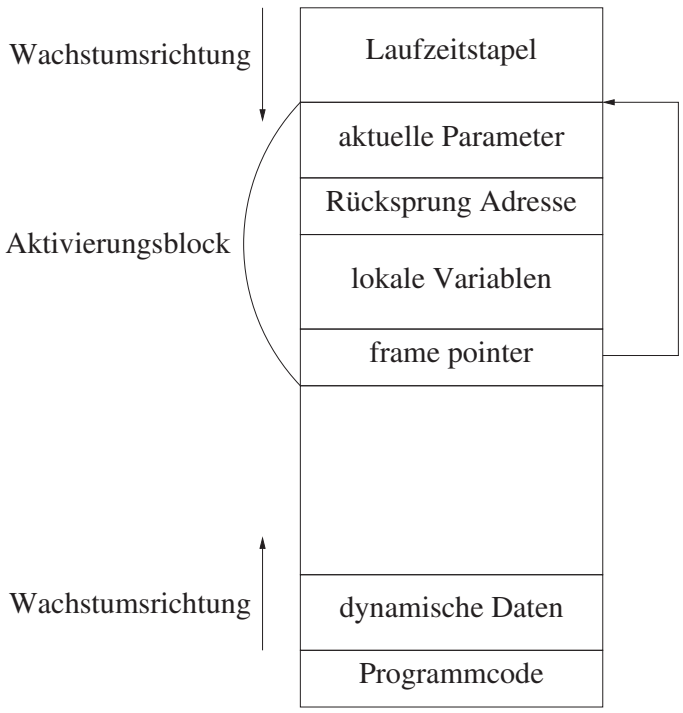

Abbildung 1.5: Hauptspeicheraufbau in prozeduralen Sprachen

2. Vom Programm zur Laufzeit *implizit* benötigte Speicherbereiche: Hierzu gehört der Speicherplatz, der beispielsweise bei einem Funktionsaufruf benötigt wird.

Da der Platzbedarf für diese Bereiche nicht im voraus fest steht, organisiert man den Speicher wie folgt: Der eine Typ von Daten wächst vom unteren Ende des freien Speichers in Richtung Mitte, während der andere Bereich vom oberen Ende Richtung Mitte wächst. Hierbei hofft man natürlich, dass zur Laufzeit des Programms diese Bereiche nicht überlappen.

Zur Laufzeit eines Programms werden Funktionen aufgerufen, in einem C-Programm beginnt der Ablauf beispielsweise mit der Funktion `main()`. Für jede aufgerufene Funktion wird ein **Aktivierungsblock** (engl. **stack frame**) auf dem Laufzeitstapel angelegt. Darin ist Platz für die aktuellen Parameter der Funktion, die Rücksprungadresse, die lokalen Variablen und einen Zeiger auf den vorherigen Aktivierungsblock. Der Zeiger auf den vorherigen Aktivierungsblock ist erforderlich, um in Sprachen wie PASCAL oder Modula 2 auf Variablen in umschließenden Prozeduren zugreifen zu können. Die Rücksprungadresse enthält die Adresse der Anweisung, mit der nach dem Ende der Funktion fortgefahren werden soll.

Aktivierungsblock

An dieser Stelle kann ein Angreifer nun versuchen, eigenen Programmcode zur Ausführung zu bringen. Dazu muss er zwei Dinge erreichen:

1. Den Programmcode, den der Angreifer ausführen möchte, muss er in den Adressraum des Programms kopieren. Hierfür bietet sich der Speicher für lokale Variablen auf dem Laufzeitstapel an.

2. Die Rücksprungadresse muss auf den Speicherbereich zeigen, in den der Angreifer seinen Code kopiert hat.

Um das zu erreichen, muss der Angreifer den Prozessortyp des anzugreifenden Systems kennen und die entsprechenden Maschinencodes eingeben können.

Unter Linux auf *Intel* Prozessoren kann man beispielsweise in unter 50 Bytes Länge Programmcode unterbringen, der eine Shell startet. Der Angreifer könnte also anschließend alle Kommandos auf dem Rechner ausführen.

Die folgenden System-/Bibliotheksfunktionen sind mögliche Ursachen für einen Speicherüberlauf. Sie sollten bei der Programmierung durch andere Funktionen ersetzt werden.

Unsichere Funktion	sicherere Alternative
gets	fgets
scanf	(Größenbegrenzung im Format-Tag)
sprintf	snprintf
strcpy	strncpy
strcat	strncat

Diese Liste ist bei weitem nicht vollständig. Viega und McGraw [VM01] haben dem Thema *Buffer Overflow* ein ganzes Kapitel gewidmet und darin eine Tabelle mit gefährlichen Funktionen, dem Grad der Gefährlichkeit und möglichen Lösungen des Sicherheitsproblems angegeben. Auch Hoglund und McGraw [HM04] haben sich ausführlich mit Buffer Overflows und wie man sie in verschiedensten Umgebungen ausnutzen kann beschäftigt.

Neben dem falschen Einsatz dieser Funktionen können Programmierer aber auch den Fehler machen, Daten in einer Schleife zu lesen und dabei die Längenprüfung „vergessen“. Ein typisches Code-Beispiel hierfür ist:

```
int     zeichen, i;
char    puffer[42];

i = 0;
while ( (zeichen = getc(stdin)) != '\n')
{
        puffer[i] = zeichen;
        ...
        i++;
}
```

Der Programmierer möchte alle Zeichen einer Zeile lesen und hat in seinem Speicherbereich aber nur Platz für 42 Zeichen. Ist die Eingabe länger, so tritt ein Buffer Overflow auf.

Gegenmaßnahmen

Buffer Overflow Fehler stecken in vielen Programmen. Die einzige Chance um diesen Fehlern zu entgehen besteht darin, die Programme entweder sorgfältiger zu schreiben oder bereits existierende Programme besonders auf diese Fehler hin zu untersuchen. Bei der Untersuchung können spezielle Hilfsprogramme die Quelltexte automatisch nach „verdächtigen“ Stellen durchsuchen und dem Programmierer einiges an Arbeit abnehmen. Die genaue Überprüfung der gefundenen Stellen und die evtl. erforderliche Korrektur bleibt aber dem menschlichen Programmentwickler vorbehalten.

Man kann Buffer Overflow Fehler auch durch die Wahl einer anderen Programmiersprache verhindern. Sprachen wir Java oder C# wurden so definiert, dass Speicherbereiche genauer kontrolliert werden und somit Buffer Overflows nicht mehr möglich sind.

1.6 URL hacking und Phishing

In diesem Abschnitt geht es um einen Angriffstyp, bei dem wieder eine falsche Identität, bzw. eine falsche Absicht erzeugt wird. Dazu werden beispielsweise kleine Fehler (Flüchtigkeitsfehler) ausgenutzt. Zu dieser Kategorie Fehler gehören Tippfehler wie Buchstaben- oder Zahlendreher. Ein Angreifer nutzt dies aus, indem er DNS-Namen registriert, die leichte Abwandlungen bekannter Namen sind. Ein Beispiel hierfür ist der Name `wwwgmx.de`.

Abbildung 1.6: Kleine Änderung, große Wirkung

Er sieht auf den ersten Blick aus wie der Name eines bekannten E-Mail Services. Allerdings fehlt der Punkt hinter dem Präfix `www`, so dass der DNS-Server hier zwei völlig verschiedene IP-Adressen liefert. Am 17. Juli 2002 waren dies:

Name	IP-Adresse	Zweck
www.gmx.de	213.165.65.100	E-Mail Service
wwwgmx.de	212.227.119.94	Kontaktanzeigen

Ein Benutzer, der häufig seine URLs beim Surfen selbst eintippt, kann somit schnell auf einem anderen System landen. Nicht immer erkennt man den Fehler so schnell wie in oben genanntem Beispiel. Hätte der Angreifer auf seinem Rechner die Website des Opfers etwas genauer nachgebaut, so könnte er erheblichen Schaden verursachen. Falls das Opfer eine Bank oder Behörde wäre, so könnte der Angreifer vertrauliche Daten erfahren. Hierzu gehört eine mögliche Benutzer-Kennung und das zugehörige Passwort.

Das gezeigte Beispiel gehört allerdings nicht in diese Kategorie. Statt dessen geht es dem Urheber in erster Linie darum, Benutzer im WWW auf seine Seite zu locken. Möglicherweise verdient er mit der dort dargestellten Werbung Geld und wird pro Besucher oder pro Seitenaufruf bezahlt.

Heute (Mai 2014) funktioniert das Beispiel aus Abbildung 1.6 nicht mehr. Betreiber großer Web-Sites sind dazu übergegangen, auch ähnliche Namen für sich zu registrieren. Sollte ein ähnlicher Name bereits registriert sein und möglicherweise mit kommerziellen Inhalten versehen sein, dann können die Inhaber des „Original“-Namens auf Unterlassung und Herausgabe des ähnlichen Namens klagen. Die Domäne `wwwgmx.de` gehört inzwischen auch zum bekannten Anbieter von E-Mail-Diensten. Eine IP-Adresse gehört damit zu mehreren DNS-Namen, die man als Synonyme für den dahinter stehenden Web-Auftritt betrachten kann. Das Online-Portal des Bayerischen Rundfunks erreicht man

z. B. unter den Namen `www.bayerischer-rundfunk.de`, `www.br-online.de` oder `www.bronline.de`.

Suchmaschinen manipulieren

Um möglichst viele Besucher auf die eigenen Seiten zu locken, gibt es weitere Möglichkeiten. Hierzu gehört die Manipulation von Suchmaschinen, so dass man bei möglichst vielen Anfragen weit oben in der Trefferliste steht. Dazu müssen bestimmte Schlüsselwörter auf der Seite vorkommen. Die Suchmaschine indiziert diese Schlüsselwörter und bewertet die Seite entsprechend. Ruft ein Benutzer die Seite im Browser dann auf, so findet er keines der Schlüsselwörter mehr, sondern etwas völlig anderes. Die möglichen Gründe hierfür sind, dass das Schlüsselwort

- nur in einem Meta-Element vorkommt, dessen Inhalt niemals vom Browser angezeigt wird,
- in weißer Schrift auf weißem Grund steht,
- in einem nicht druckbaren Bereich steht.

Eine andere Möglichkeit ist es, dass das Schlüsselwort zwar auf der Seite vorkommt, aber in einem anderen Zusammenhang benutzt wird. Schneier [Sch00] nennt hierzu folgendes Beispiel: „Dieser billige Pullover (nicht Prada, nicht Armani) ist rot.[3]“

Phishing-Angriffe

Eine Variante dieser Angriffsform sind die sogenannten **Phishing-Angriffe**. Ein Angreifer baut die HTML-Seiten, z. B. einer Bank, realistisch und echt aussehend auf seinem Server nach. Für seinen Server registriert der Angreifer einen Namen, der dem der angegriffenen Bank sehr ähnlich ist. Nun verschickt der Angreifer massenhaft E-Mail in denen sinngemäß folgendes steht:

> Lieber *Bank*–Kunde,
> wir hatten leider technische Probleme und daher müssen Sie sich leider erneut auf unserem Banking-Server registrieren. Klicken Sie bitte einfach auf die folgende URL:
> `https://banking.bamk.de/registierung/`
> Vielen Dank für Ihr Verständnis
> Ihre *Bank*

Oftmals wird die E-Mail auch als HTML verschickt, so dass auf dem Client-Rechner eine andere URL angezeigt wird als diejenige, zu der ein Klick auf die URL tatsächlich führt. Ein Beispiel:

```
<a href="http://angreifer.de"> http://www.opfer.de </a>
```

In HTML ist es der Wert des Attributs `href`, der angibt wohin ein Klick des Benutzer denn führt. Der Inhalt des Elements `<a>` (zwischen Start tag und End tag) ist das, was dem Benutzer auf der HTML-Seite angezeigt wird. Auch der Einsatz von Kodierungstechniken für URLs kann dazu führen, die URL für den Menschen schwer überprüfbar zu machen.

Deshalb empfiehlt es sich, keine URLs aus verdächtigen E-Mails direkt anzuklicken. Statt dessen sollte man die Adresse seiner Bank in die Favoriten-Liste des Browsers eintragen und immer nur über die Liste anklicken. Außerdem schadet es auch nicht, hin und wieder das SSL-Zertifikat zu überprüfen.

[3] Prada und Armani sind Markennamen von Kleidungsstücken.

1.7 Gefährliche Benutzereingaben

Viele Rechner sind gefährdet, wenn sie Benutzereingaben übernehmen und diese dann ungeprüft an weitere Programme übergeben. Böswillige Benutzer könnten dann Eingaben machen, die die Programmierer so nicht erwartet haben. Wenn diese Eingaben dann verarbeitet werden, können Sicherheitsprobleme entstehen. In Abschnitt 1.7.1 geht es darum, dass manche Benutzereingaben später in HTML-Seiten dynamisch eingefügt werden. Diese Seiten werden dann von anderen Benutzern geladen, d. h. vom Browser des anderen Benutzers, und interpretiert. Dabei können dann unerwünschte Effekte auftreten.

Abschnitt 1.7.2 diskutiert das Problem, wenn Benutzereingaben direkt an eine Datenbank weitergeleitet werden. In vielen Web-Anwendungen werden SQL-Anweisungen direkt anhand der Benutzereingaben zusammengestellt und dann ausgeführt. Mit passenden Benutzereingaben kann ein Angreifer unbefugt die Datenbank manipulieren.

1.7.1 HTML Code

In HTML-Seiten können aktive Inhalte integriert sein. Diese aktiven Inhalte bergen einige Sicherheitsrisiken; sie sind z. B. in Kurs *(01866) Sicherheit im Internet 1* beschrieben. Das können sich Angreifer zunutze machen, wenn sie Eingaben erzeugen dürfen, die dann später in HTML-Seiten eingebettet sind.

Ein typisches Beispiel hierfür sind Gästebücher auf persönlichen Web-Sites. Dort kann jedermann in einem HTML-Formular Einträge für das Gästebuch verfassen. Schaut sich ein anderer Benutzer den Inhalt des Gästebuches an, dann werden die Eingaben der anderen Gäste in eine HTML-Seite eingebettet und dem aktuellen Besucher angezeigt. Hat der Benutzer in ein Formular beispielsweise eingegeben:

```
Lieber Homepage Betreiber,

ich finde deine Seiten total toll und werde sie
allen meinen Freunden weiterempfehlen.

Viele Gruesse
Der Kritiker
```

Daraus kann dann bei Abruf der Seite der folgende HTML-Code generiert werden:

```
<html>
<head>
... der typische HTML Kopf ...
</head>
<body>
<h1>Grosses Gaestebuch von toller Hecht</h1>
Der Benutzer Kritiker schrieb am 1.4.2005 ueber
diese Seiten:
<pre>
Lieber Homepage Betreiber,

ich finde deine Seiten total toll und werde sie
```

allen meinen Freunden weiterempfehlen.

Viele Gruesse
Der Kritiker
</pre>
... hier kommen weitere Kritiken oder was auch immer ...
</body>

Hat ein vorheriger Besucher in der Eingabemaske für seinen Eintrag nun HTML-Elemente eingetippt, dann werden diese HTML-Elemente vom Browser des nachfolgenden Benutzers interpretiert und angezeigt. Ein böswilliger Benutzer könnte also `<script>`-, `<applet>`- oder ähnliche HTML-Elemente eintippen, die dann ausgeführt werden und dabei möglicherweise Schäden verursachen.

1.7.2 SQL injection

In vielen E-Commerce Anwendungen müssen Daten aus Datenbanken abgerufen werden. Meldet sich beispielsweise ein Benutzer an, dann muss die Benutzer-Kennung in einer Datenbank nachgeschlagen werden. Eventuell wird sogar zusätzlich noch ein Passwort überprüft. Aber auch an anderen Stellen sind Datenbankenzugriffe erforderlich. Beispiele hierfür sind: Warenauswahl aus einem Katalog, Verfügbarkeitsüberprüfungen, etc.

Structured Query Language (SQL)

Als Sprache für die Datenbankabfragen (engl. **query**) wird meistens **Structured Query Language (SQL)** benutzt. Weitere Details zum Thema Datenbanken und SQL werden in Kurs *(01671) Datenbanksysteme 1* vorgestellt. In SQL kann man neben Anfragen (engl. **query**) aber auch Manipulationen an einer Datenbank vornehmen. Eine einfache SQL-Anfrage sieht beispielsweise so aus:

```
select * from user where kennung = 'stefan';
```

Diese Anweisung sucht in der Tabelle `user` nach allen Zeilen, die in der Spalte `kennung` die Zeichenkette `stefan` enthalten. Die Werte aller Spalten dieser Zeilen werden dann ausgegeben.

Wichtig für das Verständnis von SQL injection sind die folgenden drei Konzepte von SQL:

1. Zeichenketten (engl. **strings**) werden in einfache Anführungszeichen gesetzt.
2. Mehrere SQL-Anweisungen werden durch ein Semikolon getrennt.
3. Kommentare in SQL-Anweisungen beginnen mit `--`. Der Rest der Zeile wird dann vom SQL-Interpreter ignoriert.

In Internet-Anwendungen möchte man dem Benutzer erlauben, die Zeichenketten selbst einzugeben. Die Anwendung baut dann aus den Eingaben die SQL-Anweisung zusammen. Steht die Benutzereingabe beispielsweise in einer Variablen `eingabe`, dann kann man in einem Java-Programm durch die folgende Zeile eine SQL-Anweisung konstruieren:

```
String anw = new String("select * from user where
            kennung = '" + eingabe + "';");
```

Der Java-Operator + konkateniert Strings, die in Java in doppelte Anführungszeichen gesetzt sind. Hat die Variable `eingabe` den Wert `stefan`, dann ensteht in der Variablen `anw` die oben bereits gezeigte SQL-Anweisung. Um diese Anweisung tatsächlich auszuführen, braucht es in **Java Database Connectivity (JDBC)** noch eine Verbindung zur Datenbank und ein Statement-Objekt. Der Java-Code sieht dann so aus:

Java Database Connectivity (JDBC)

```
import java.sql.*;

// Sei getConnection eine Methode, die
// die Verbindung zur Datenbank aufbaut.

Connection conn = getConnection();
Statement  stat = conn.getStatement();
String     anw = new String( /* siehe oben */ );

ResultSet rs = stat.executeQuery(anw);

// Nun noch die Ergebnismenge verarbeiten
```

Wenn der Benutzer in die Variable `eingabe` aber folgendes eingibt, entsteht eine neue SQL-Anweisung. Aus der Eingabe:

```
'; drop table user;--
```

wird diese SQL-Anweisung:

```
select * from user where kennung = ''; drop table user; --';
```

Diese Anweisung ist eine Sequenz aus zwei Anweisungen. Die erste durchsucht die Tabelle `user` und die zweite Anweisung (`drop table user;`) löscht die Tabelle `user` aus der Datenbank.

Die Ursache dieses Angriffs liegt darin, dass der Benutzer ein einfaches Hochkomma eingeben konnte und damit die ursprüngliche SQL-Anweisung „vorzeitig“ beenden kann. Der Rest der Eingabe wird als neue SQL-Anweisung interpretiert. Man kann dort alle SQL-Anweisungen eingeben und damit beliebige Manipulationen an der Datenbank vornehmen. Dazu ist es allerdings erforderlich, die Namen der Tabellen und der Attribute (Spalten) zu kennen. Kennt man sie nicht, dann müsste ein Angreifer die Namen raten. Voraussichtlich wird er dabei erst einmal falsche Namen ausprobieren. Je nach Datenbanksystem werden dabei verschieden ausführliche Fehlermeldungen erzeugt. Diese Fehlermeldungen enthalten dann möglicherweise die bisher unbekannten Tabellen- oder Attribut-Namen.

Um diese Angriffe zu verhindern, müssen Anwendungen den Benutzereingaben *immer* misstrauen. Jede Eingabe muss von der Anwendung überprüft werden. Hierzu können zwei Strategien eingesetzt werden:

Gegenmaßnahmen

1. Durchsuche die Eingabe nach „gefährlichen“ Zeichen und ersetze sie durch ungefährliche Alternativen.

2. Lasse in der Eingabe nur bestimmte Zeichen (oder reguläre Ausdrücke) zu.

Die erste Variante hat den Nachteil, dass man beim Entwurf der Prüfung Zeichen für ungefährlich hält, die sich später aber evtl. als gefährlich herausstellen. Dann wäre die Überprüfung unvollständig und muss geändert werden. Die zweite Variante ist sicherer, da sie die ungefährlichen Eingaben beschreibt und alles Abweichende abweist.

Java-Programmierer, die den Datenbankzugriff mit JDBC implementieren, können an Stelle eines *Statement*-Objekts auch ein sogenanntes *Prepared-Statement* (vorbereitete Anweisung) benutzen. In einem PreparedStatement dürfen nur an bestimmten Stellen die Benutzereingaben einkopiert werden. Das Datenbanksystem kann bei einem PreparedStatement vorab die günstigste Ausführungsreihenfolge planen. Benutzt man das PreparedStatement dann mehrmals (immer mit anderen Parametern), so spart die Datenbank bei den Folgeaufrufen die Planung und kann die Anfrage so schneller beantworten. Außerdem kann das PreparedStatement-Objekt prüfen, ob die Benutzereingaben die Struktur der Anweisung verändern oder nicht. Der Java-Code sieht dann so aus:

```
import java.sql.*;

// Sei getConnection eine Methode, die
// die Verbindung zur Datenbank aufbaut.

Connection conn = getConnection();

String anw = new String("SELECT * FROM user" +
             " WHERE kennung = ?" );

PreparedStatement pstat = conn.prepareStatement(anw);

// Der String eingabe enthalte die Benutzereingabe
// Setze den 1. Parameter (? im Anweisungs-String)
// auf den Wert der Variablen eingabe
   pstat.setString(1, eingabe);

ResultSet rs = pstat.executeQuery();

// Und wieder die Ergebnismenge verarbeiten
```

Der String für das PreparedStatement enthält an den Stellen, an denen später noch Werte gesetzt werden sollen, nur ein Fragezeichen. Die set-Methoden des PreparedStatement-Objekts können nun den Wert einer Variablen an Stelle eines Fragezeichens einfügen. Und setString kann beispielsweise prüfen, dass der Wert tatsächlich ein „normaler“ String ist, der kein einfaches Anführungszeichen enthält. Weitere Details zu JDBC finden Sie beispielsweise in [HC05].

1.8 Spezielle „Hacker“ Tools

Es existieren verschiedene Hilfsprogramme, mit denen die Sicherheit eines Systems getestet werden kann. Hierbei kann man drei wesentliche Klassen unterscheiden:

1. **Host Scanner** untersuchen einen einzelnen Rechner und testen beispielsweise die Rechte-Vergabe, die Konfiguration sowie die Stärke der Passwörter. Host Scanner
2. **Network Scanner** untersuchen ein Netz und konzentrieren sich auf die Kommunikation zwischen Rechnern. Network Scanner
3. **Intrusion Scanner** versuchen einen Einbruch in einen Rechner oder ein Netz zu erkennen. Sie werden später im Kurs behandelt (siehe Abschnitt 3.3). Intrusion Scanner

Mit der Kali-Linux Distribution existiert ein freies Betriebssystem, bei dem die aktuellen Versionen von vielen frei verfügbaren „Hacker“ Tools bereits installiert sind. Man spart sich die Suche und das Herunterladen der einzelnen Programme. Außerdem kann man die Distribution auch als virtuelle Maschine starten und damit herumspielen. Man findet sie im Internet unter der URL `http://www.kali.org/`. In den beiden folgenden Unterabschnitten werden die beiden erstgenannten Klassen vorgestellt.

1.8.1 Host Scanner

Ein Host Scanner untersucht die Konfiguration eines Rechners auf Schwachstellen. Das war früher deutlich wichtiger als heute, weil einige Betriebssystemhersteller ihre Systeme im Auslieferungszustand unsicher konfiguriert hatten. So wurden beispielsweise viele Dienste aktiviert (der Benutzer könnte sie ja brauchen), obwohl die meisten Benutzer diese Dienste nicht benutzen wollten und gar nicht wussten, dass dieser Dienst aktiviert war. Angreifer fanden darüber aber einen Startpunkt. Heute sind die meisten Hersteller vorsichtiger und liefern ihre Systeme in einer sicheren Grundkonfiguration. Der Administrator muss Änderungen selbst vornehmen und sollte dabei dann natürlich wissen was er macht. Host Scanner untersuchen die folgenden Punkte: Problembereiche

Zugriffsrechte von Dateien: Ein typisches Problem, das hier auftreten kann, sind Leserechte für zu viele Benutzer. Ein Wörterbuchangriff (siehe Abschnitt 1.4) benötigt Zugriff auf die Passwort-Datei. Falls diese Datei mit Leserechten für alle Benutzer versehen ist, kann dieser Angriff Erfolg haben.

Besitzer von Dateien: Falls eine Datei einem falschen Besitzer oder einer falschen Gruppe zugeordnet ist, kann ein Angreifer diese Datei unberechtigt lesen oder verändern. Ausführbare Dateien mit dem SUID-Bit laufen mit den Rechten des Besitzers der Datei, nicht mit den Rechten desjenigen, der das Programm startet (siehe auch Kurs *(01801) Betriebssysteme und Rechnernetze*). So kann ein normaler Benutzer auf die Passwort-Datei zugreifen (z. B. um sein Passwort zu ändern), ohne selbst die Zugriffsrechte zu besitzen. Statt dessen ist das Programm, das die Passwort-Datei liest und verändert, mit entsprechenden Rechten ausgestattet. Das Programm gehört dem Administrator und läuft daher mit dessen Rechten.

Ein Angreifer könnte nun versuchen, sein Programm mit gesetztem SUID-Bit dem Benutzer `root` zuzuordnen. Dieses Programm enthält eine Schadens-Funktion und würde, egal von welchem Benutzer es gestartet wird, immer die Zugriffsrechte von `root` haben und könnte die Schäden dann auch immer anrichten.

Unbefugte Modifikationen: Ein Host Scanner kann auch die Prüfung der Integrität von Daten vornehmen. Mit Hilfe von digitalen Signaturen können Veränderungen entdeckt und evtl. auch rückgängig gemacht werden.

Inhalte von Konfigurationsdateien: Falls auf dem System spezielle Systemsoftware, beispielsweise ein Datenbanksystem installiert wurde, so enthält die zugehörige Konfiguration häufig einige Probleme. Es können Standard-Benutzer mit bekannten Standard-Passwörtern vorkonfiguriert sein. Diese Konfiguration sollte auf einem produktiven System natürlich geändert sein. Bei Aktualisierungen (engl. **update**) der Software sollte die Konfiguration jedes mal mit geprüft werden.

Versionen der Software: In vielen Programmen wurden Sicherheitsprobleme festgestellt. Diese wurden in aktuelleren Versionen behoben. Ein Host Scanner kann prüfen, ob Programme mit bekannten Schwachstellen auf dem System installiert sind.

Im folgenden werden die Host Scanner *lynis* für Linux und *MBSA* für MS Windows vorgestellt.

lynis: Für UNIX-Systeme gibt es mit *lynis* ein freies Werkzeug mit dem man ein System überprüfen kann. Man findet das System unter der URL

`http://cisofy.com/lynis`

lynis startet eine Reihe von Sicherheitsprüfungskripten, die feststellen, welche Software-Pakete installiert sind und prüfen, ob einige Sicherheitsrichtlinien eingehalten werden. Auch werden einige Konfigurationsdateien auf sicherheitskritische Einstellungen hin überprüft.

Das Ergebnisprotokoll eines lynis-Laufs ist sehr umfangreich. Neben den Analyseergebnissen enthält er auch Hinweise (engl. **suggestion**) oder Warnungen (engl. **warning**). Diese sollte der Benutzer besonders aufmerksam lesen und ggf. seine Konsequenzen ziehen. Eine gekürzte (die komplette Datei hat ca. 1800 Zeilen) Ausgabe von lynis kann wie folgt aussehen:

```
# Lynis Report
report_version_major=1
report_version_minor=0
report_datetime_start=2014-03-17 20:13:16
auditor=[Unknown]
lynis_version=1.4.6
os=Linux
os_name=Debian
os_fullname=Debian 6.0.9
os_version=6.0.9
linux_version=Debian
hostname=debian6
lynis_update_available=0
binaries_count=2671
exception[]=No eth0 found (but HWaddr was found), using first network interface to determine hostid
hostid=ae234ebf8503832095bf77fa55ce1d1a95acd26c
manual_event[]=BOOT-5121:01
boot_loader=GRUB2
linux_default_runlevel=2
cpu_pae=1
cpu_nx=1
linux_kernel_release=2.6.32-5-amd64
linux_kernel_version=#1 SMP Mon Sep 23 22:14:43 UTC 2013
linux_kernel_type=modular
loaded_kernel_module[]=ata_generic
loaded_kernel_module[]=ata_piix

...

loaded_kernel_module[]=virtio_pci
loaded_kernel_module[]=virtio_ring
loaded_kernel_module[]=x_tables
memory_size=2061084
memory_units=kB
real_user[]=root,0
```

```
real_user[]=stefan,1000
real_user[]=ossec,50731
real_user[]=ossecm,50732
real_user[]=ossecr,50733
suggestion[]=AUTH-9262|Install a PAM module for password strength testing like pam_cracklib or pam_passwdqc|
suggestion[]=AUTH-9286|Configure password aging limits to enforce password changing on a regular base|
exception_event[]=AUTH-9328:01
manual_event[]=AUTH-9328:01
manual_event[]=AUTH-9328:03
suggestion[]=AUTH-9328|Default umask in /etc/login.defs could be more strict like 027|
suggestion[]=AUTH-9328|Default umask in /etc/init.d/rc could be more strict like 027|
ldap_auth_enabled=1
ldap_pam_enabled=1
available_shell[]=/bin/csh
available_shell[]=/bin/sh

...

available_shell[]=/bin/bash
available_shell[]=/bin/rbash
suggestion[]=FILE-6310|To decrease the impact of a full /home file system, place /home on a separated partition|
suggestion[]=FILE-6310|To decrease the impact of a full /tmp file system, place /tmp on a separated partition|
locate_db=/var/cache/locate/locatedb
suggestion[]=STRG-1840|Disable drivers like USB storage when not used, to prevent unauthorized storage or data theft|
suggestion[]=STRG-1846|Disable drivers like firewire storage when not used, to prevent unauthorized storage or data theft|
resolv_conf_domain=,100024,1,tcp,51561,status
resolv_conf_search_domain[]=inform.fh-hannover.de
suggestion[]=NAME-4406|Split resolving between localhost and the hostname of the system|
package_manager[]=rpm
package_manager[]=dpkg
installed_package[]=acl|2.2.49-4|
installed_package[]=acpi|1.5-2|
installed_package[]=xz-utils|5.0.0-2|

...

installed_package[]=yelp|2.30.1+webkit-1|
installed_package[]=zenity|2.30.0-1|
installed_package[]=zlib1g|1:1.2.3.4.dfsg-3|
installed_packages=1554
suggestion[]=PKGS-7346|Purge old/removed packages (1 found) with aptitude purge or dpkg --purge command. This will cleanup old configuration files, ...
pkg_audit_tool=apt-check
pkg_audit_tool_found=1
nameserver[]=141.71.30.1
nameserver[]=141.71.30.11
default_gateway[]=141.71.31.254
network_mac_address[]=16:89:17:d3:3b:cf
network_mac_address[]=52:54:00:5a:6c:59
network_ipv4_address[]=141.71.31.218
network_ipv4_address[]=127.0.0.1
network_ipv6_address[]=fe80::5054:ff:fe5a:6c59/64
network_ipv6_address[]=::1/128
network_listen_port=0.0.0.0:68|udp|dhclient|
network_listen_port=0.0.0.0:715|udp|rpc.statd|
network_listen_port=0.0.0.0:5353|udp|avahi-daemon:|
network_listen_port=0.0.0.0:40299|udp|rpc.statd|
network_listen_port=0.0.0.0:111|udp|portmap|
network_listen_port=0.0.0.0:37746|udp|avahi-daemon:|
network_listen_port=0.0.0.0:631|udp|cupsd|
network_listen_port=:::47303|udp6|avahi-daemon:|
network_listen_port=:::5353|udp6|avahi-daemon:|
dhcp_client_running=1
suggestion[]=PRNT-2307|Access to CUPS configuration could be more strict.|
warning[]=FIRE-4512|iptables module(s) loaded, but no rules active|
suggestion[]=FIRE-4512|Disable iptables kernel module if not used or make sure rules are being used|
manual[]=Verify if there is a formal process for testing and applying firewall rules
manual[]=verify all traffic is filtered the right way between the different security zones
manual[]=verify if a list is available with all required services
manual[]=Make sure an explicit deny all is the default policy for all unmatched traffic
firewall_installed=1
firewall_active=1
firewall_software=iptables
warning[]=SSH-7412|Root can directly login via SSH|
ssh_daemon_running=1
log_directory[]=/var/log

...

open_logfile[]=/var/log/openvas/openvasmd.log
open_logfile[]=/var/log/syslog
open_logfile[]=/var/log/user.log
log_rotation_config_found=1
log_rotation_tool=logrotate
suggestion[]=BANN-7126|Add a legal banner to /etc/issue, to warn unauthorized users|
suggestion[]=BANN-7130|Add legal banner to /etc/issue.net, to warn unauthorized users|
cronjob[]=17,*,*,*,*,root,cd,/,&&,run-parts,--report,/etc/cron.hourly
cronjob[]=25,6,*,*,*,root,test,-x,/usr/sbin/anacron,||,(,cd,/,&&,run-parts,--report,/etc/cron.daily,)
cronjob[]=47,6,*,*,7,root,test,-x,/usr/sbin/anacron,||,(,cd,/,&&,run-parts,--report,/etc/cron.weekly,)
cronjob[]=52,6,1,*,*,root,test,-x,/usr/sbin/anacron,||,(,cd,/,&&,run-parts,--report,/etc/cron.monthly,)
cronjob[]=/etc/cron.daily/locate
cronjob[]=/etc/cron.daily/exim4-base

...

cronjob[]=/etc/cron.weekly/0anacron
cronjob[]=/etc/cron.monthly/0anacron
cronjob[]=1,5,cron.daily,nice,run-parts,--report,/etc/cron.daily
cronjob[]=7,10,cron.weekly,nice,run-parts,--report,/etc/cron.weekly
cronjob[]=@monthly,15,cron.monthly,nice,run-parts,--report,/etc/cron.monthly
suggestion[]=ACCT-9626|Enable sysstat to collect accounting (no results)|
suggestion[]=ACCT-9628|Enable auditd to collect audit information|
audit_deamon_running=0
suggestion[]=TIME-3104|Use NTP daemon or NTP client to prevent time issues.|
```

```
ntp_config_found=0
ntp_config_type_daemon=0
ntp_config_type_eventbased=0
ntp_config_type_scheduled=0
ntp_config_type_startup=0
ntp_daemon=
ntp_daemon_running=0
expired_certificate[]=/etc/ssl/certs/ca-certificates.crt
warning[]=CRYP-7902|One or more SSL certificates expired|
suggestion[]=FINT-4350|Install a file integrity tool|
file_integrity_installed=0
malware_scanner_installed=0
warning[]=FILE-7524|Incorrect permissions for file /root/.ssh|
home_directory[]=/bin
home_directory[]=/dev
home_directory[]=/home/stefan
home_directory[]=/root
home_directory[]=/usr/games

...

home_directory[]=/var/run/sshd
home_directory[]=/var/run/stunnel4
home_directory[]=/var/spool/exim4
suggestion[]=KRNL-6000|One or more sysctl values differ from the scan profile and could be tweaked|
suggestion[]=HRDN-7220|Harden the system by removing unneeded compilers. This can decrease the chance of customized trojans, ...
suggestion[]=HRDN-7222|Harden compilers and restrict access to world|
suggestion[]=HRDN-7230|Harden the system by installing one or malware scanners to perform periodic file system scans|
compiler_installed=1
lynis_tests_done=163
report_datetime_end=2014-03-17 20:16:55
hardening_index=57
tests_executed=HRDN-7230|HRDN-7222|HRDN-7220|KRNL-6000|HOME-9350|HOME-9310|HOME-9302|FILE-7524|MALW-3284|MALW-3282|MALW-3276|...
tests_skipped=MALW-3286|MACF-6234|MACF-6208|VIRT-1902|TIME-3136|TIME-3132|TIME-3128|TIME-3124|TIME-3120|TIME-3116|TIME-3112|...
finish=true
```

Am Anfang kommen einige allgemeine Informationen über die Versionsnummer von *lynis* und den Host, der untersucht wird. Im Beispiel oben handelt es sich um einen Test-Server mit dem Betriebssystem Debian in Version 6. Nach der Liste der Kernel-Module kommen die lokalen Benutzer und der Vorschlag, ein Modul zu installieren, der die Stärke von Benutzerpasswörtern prüft.

Den größten Teil der Ausgabe besteht aus der Liste der installierten Software-Pakete. Ihr folgt der Abschnitt über die Netzeigenschaften des Systems, wie IP-Konfiguration, angebotene Dienste, usw. Auf dem Server ist beispielsweise keine Firewall eingerichtet und dementsprechend gibt es den Vorschlag, doch *iptables* zu benutzen. Im Abschnitt cronjobs werden die regelmäßig und automatisch von System gestarteten Programme angezeigt. Danach kommen Hinweise zu möglichen Problemen mit dem Network Time Protocol (NTP) und einigen abgelaufenen Zertifikaten.

Mit *lynis* bekommt man ein freies Werkzeug, mit dem man bequem einen UNIX-Rechner untersuchen kann. Möchte man regelmäßige Überprüfungen (engl. **audits**) durchführen und hat man viele Rechner, die überprüft werden sollen, so wird es mit der freien Version schwierig. Die Autoren von lynis bieten eine kommerzielle Version an, die diese erweiterten Anforderungen abdeckt.

Microsoft Baseline Security Analyser (MBSA): Dieses Programm kann man kostenlos von den Webseiten der Firma *Microsoft* herunterladen. Es dient im wesentlichen dazu, den Sicherheitsstatus von Rechnern mit einem Betriebssystem der Windows-Familie zu prüfen. Konkret geht es um:

- Überprüfung der Konfiguration eines Rechners, beispielsweise auf eingeschaltete Dienste, Laufwerks-Freigaben, Sicherheit der Passwörter gegen brute force Angriffe, automatische Anmeldung, usw.

- Überprüfung, ob auch alle Sicherheitsaktualisierungen (engl. **security updates**) für das Betriebssystem installiert sind. Dazu muss sich das Programm natürlich mit einem Server bei *Microsoft* verbinden.

- Überprüfung weiterer Programme von Microsoft (z. B. der Web-Server *Internet Information Server*, der Datenbank-Server *MS SQL Server*, usw.), falls sie installiert sind.

Hinweise zum Microsoft Baseline Security Analyser finden Sie im Internet unter der URL:

`http://technet.microsoft.com/de-de/security/cc184923.aspx`

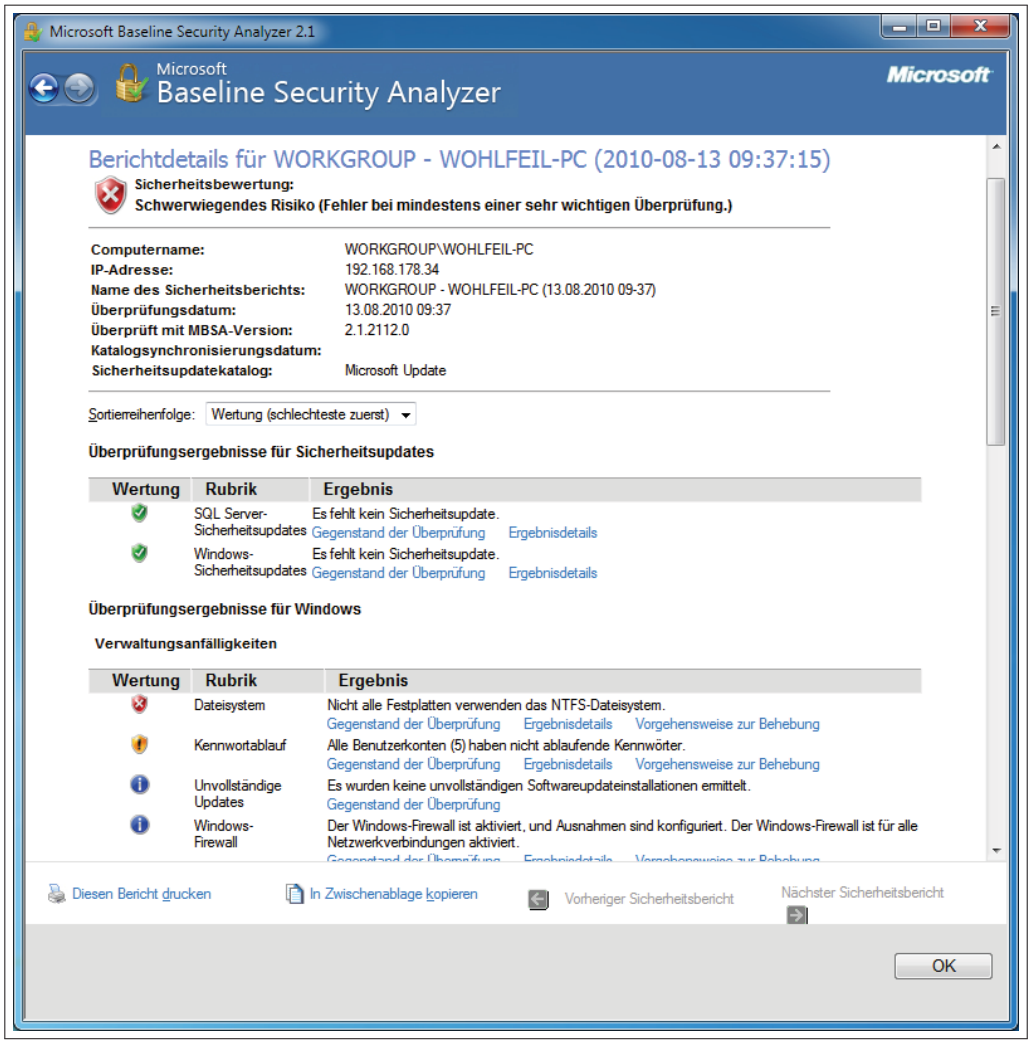

Abbildung 1.7: Ausgabe von *MBSA*

In Abbildung 1.7 ist eine Beispielausgabe von *MBSA* dargestellt. Das einzige schwerwiegende Sicherheitsproblem auf einem Windows 7 Laptop ist eine FAT-Partition. Der Grund ist, dass auf FAT-Partitionen keine Dateizugriffsrechte verwaltet werden können. Eine solche Partition eignet sich also nur als Austauschpartition. Hat man mehrere Betriebssysteme auf einem Rechner installiert, so kann man sich immer sicher sein, dass jedes Betriebssystem diese FAT-Partition lesen und schreiben kann. Außerdem hat MBSA festgestellt, dass die Passwörter niemals ablaufen. Die Möglichkeit, dass Benutzer regelmäßig ihre Passwörter ändern müssen, ist deaktiviert.

Neben der Überprüfung des Rechners, auf dem es läuft, kann der MBSA auch andere Rechner im Netz überprüfen. Das leitet direkt zum nächsten Abschnitt über.

Kurs 01867 – Sicherheit im Internet 2
Einsendeaufgaben Kurseinheit 1

Aufgabe 1: (15 Punkte)

Welche vertraulichen Daten kann man durch einen Sniffer erfahren, wenn der angegriffene Benutzer die folgenden Protokolle verwendet:

a) telnet

b) HTTP

c) POP3

Aufgabe 2: (20 Punkte)

Da man Passwörter ja nicht aus einem Wörterbuch nehmen darf, hat Peter Pfiffig die Idee, einfach zwei Wörter aus dem Wörterbuch zu benutzen. Er bildet das Passwort nun, indem er ein Wort, eine Ziffer und dahinter das zweite Wort schreibt (zum Beispiel: Klaus7Luise). Im Wörterbuch stehen n verschiedene Wörter. Wieviel Aufwand muß ein Angreifer im Mittel treiben, um das Passwort von Peter Pfiffig durch einen brute force Angriff zu ermitteln?

Aufgabe 3: (15 Punkte)

Zerlegen Sie mit Hilfe des Quadratic Sieve Algorithmus die Zahl 703 in ihre Primfaktoren. Welchen Wert haben die Zahlen x und y?

Aufgabe 4: (30 Punkte)

Kreuzen Sie bei den folgenden Aussagen an, ob sie richtig/wahr (R) oder falsch (F) sind. Für jedes richtige Kreuz gibt es drei Punkte, für jedes falsche Kreuz werden drei Punkte abgezogen. Sollte Ihre Punktzahl negativ werden, dann wird diese Aufgabe mit Null Punkten gewertet.

Aussage	R	F
Ein Ethernet-Hub sendet einen Ethernet-Frame nur auf den Anschluß, an dem das Gerät mit der Empfänger-MAC-Adresse eingestöpselt ist.		
Bei einem Sniffing-Angriff auf einen FTP-Server können Sie Benutzerkennungen und Passwörter im Klartext ausspionieren.		
Ein Angreifer kann Ihren PC mit DNS-Spoofing angreifen, ohne einen DNS-Server manipulieren zu müssen.		
Ein Passwort der Länge acht, das nur aus Großbuchstaben (von den 26 Buchstaben des Lateinischen Alphabets) und Ziffern besteht ist aus einer Gesamtmenge der Größe 8^{26+10}.		
Begrenzt man die Länge eines Eingabefeldes in einem HTML-Fomular auf 50 Zeichen, so kann mit den Eingaben dieses Eingabefeldes kein SQL-Injection-Angriff mehr durchgeführt werden.		
Erlaubt man in einem Eingabefeld ausschließlich ASCII-Zeichen, so ist ein SQL-Injection-Angriff mit den Eingaben aus diesem Feld trotzdem möglich.		
Hat ein Angreifer eine Chip-Karte eines Opfers in seinem Besitz, so kann er mit einem Adaptive Chosen Plaintext Angriff versuchen, den geschützten Schlüssel auf der Karte zu „knacken“.		
Benutzt man als Java-Programmierer die JDBC-Schnittstelle zum Datenbankzugriff und darin ein normales Statement-Objekt, so ist man vor SQL-Angriffen sicher.		
Mit einem Rootkit sorgt ein Angreifer dafür, dass er auch in Zukunft Administrator-Zugriff auf den angegriffenen Rechner hat.		
Nachdem ein Kernel-based-Rootkit auf einem Rechner installiert wurde, kann man sich auf die Ausgaben eines Anti-Viren-Programms nicht mehr verlassen.		

Prof. Dr. Armin R. Mikler

Kurs 21811 Fehlertoleranz in Computersystemen und Netzwerken

LESEPROBE

mathematik und informatik

FernUniversität in Hagen

Vorwort

Dieser Kurs besteht aus vier Kurseinheiten, die eine Einführung in das Gebiet der fehlertolerierenden Systeme vermitteln. Die erste Einheit macht den Leser durch eine Reihe von Beispielen zunächst mit dem Problem der Fehlertoleranz bekannt. Darüber hinaus sollen in der ersten Kurseinheit die Unterschiede zwischen Fehlertoleranz und Risikominimierung genauer untersucht werden. Obwohl das Gebiet der fehlertolerierenden Systeme im Allgemeinen im Rahmen der Informatik definiert ist, werden die grundlegenden Prinzipien anhand allgemeiner Beispiele dargestellt. Insbesondere werden hier verschiedene Arten von Fehlern definiert und untersucht, unter welchen Bedingungen ein System trotz Auftretens solcher Fehler weiterarbeiten kann. In der zweiten Kurseinheit werden die theoretischen Grundlagen behandelt, die im Wesentlichen ein Basiswissen der Wahrscheinlichkeitsrechnung und der Statistik umfassen. Hierbei werden die Prinzipien jeweils mittels Beispielen demonstriert, die es ermöglichen die behandelten Methoden auf verschiedene Probleme anzuwenden. In der dritten Kurseinheit werden Ansätze vorgestellt, die zu einer fehlertolerierenden Datenspeicherung und Datenübertragung beitragen. Dabei werden im Speziellen die Unterschiede zwischen Fehlererkennung und Fehlerkorrektur demonstriert. Darüber hinaus wird in der dritten Kurseinheit der Begriff der Redundanz genauer untersucht. Die vierte Kurseinheit fokussiert auf die Entwicklung fehlertoleranter Kommunikationssysteme und verteilte Rechnersysteme mittels protokollbasierter Ansätze. Verschiedene Protokollelemente werden hier auf ihren Beitrag zur Fehlertoleranz hin untersucht. Dabei dient das 7-Schichten OSI Modell dazu die unterschiedlichen Protokollelemente den jeweiligen Systemkomponenten zuzuordenen.

Am Ende jeder Kurseinheit findet der Leser eine Sammlung von Übungsaufgaben, die dazu beitragen einen Bezug zu der oft sehr abstrakten Materie der fehlertoleranten Systeme herzustellen. Zur Lösung dieser Aufgaben muss der Leser die jeweilige Kurseinheit sorgfältig bearbeiten und möglicherweise einen Blick in die verfügbare Literatur werfen.

Die Themen der einzelnen Kurseinheiten basieren auf aktuellen Fachbüchern der Fehlertoleranz, Kommunikationssysteme, Betriebssysteme, verteilte Systeme und der Wahrscheinlichkeitsrechnung und Statistik. Die hier vorgestellte Materie wird im Allgemeinen von allen Fachbüchern dieser Gebiete abgedeckt. Darüber hinaus bietet es sich an, einige spezielle Journalartikel zu konsultieren, die zu einem die historische Entwicklung der fehlertoleranten Systeme darstellen und zum anderen den aktuellen Forschungsaufwand darstellen.

Inhaltsverzeichnis

Kapitel 1

Prinzipien der Fehlertoleranz und Ausfallsicherheit

1.1 Lernziel der Kurseinheit

In dieser Kurseinheit sollen die Begriffe der Fehlertoleranz und der Risikominimierung definiert werden und darüber hinaus die folgenden Konzepte vorgestellt werden:

- Kategorien von Fehlern und Defekten,
- Redundanz,
- Kosten der Fehlertoleranz,
- Ausfallsicherheit und Verfügbarkeit.

Anhand verschiedener Beispiele, die nicht exklusiv aus der Informatik stammen, werden diese Konzepte genauer untersucht. Dabei dienen die am Ende des Kapitels gestellten Übungsaufgaben dazu das Verständnis der vorgestellten Konzepte selbst zu überprüfen. Im Folgenden werden verschiedene Konzepte vorgestellt, die in der zweiten Kurseinheit aufgegriffen und genauer behandelt werden.

1.2 Fehlertoleranz, Risiko und Sicherheit

Täglich treffen wir Entscheidungen, an denen sich unsere persönliche Einstellung zu möglichen Fehlern und unsere Risikobereitschaft widerspiegeln. In diesem Kapitel werden wir versuchen solche Konzepte etwas präziser zu definieren und Entscheidungen oder Verhaltensweisen genauer zu untersuchen. Zwar sind die Begriffe der Fehlertoleranz und der Systemsicherheit meist im Rahmen der Informatik definiert, dennoch ist es nützlich und hilfreich diese Konzepte im Kontext des täglichen Lebens zu untersuchen und zu verstehen. Während die oben aufgeführten Konzepte prinzipiell verschieden sind, ist es oft problematisch sie in expliziten Fällen zu unterscheiden und zu differenzieren. So ist zum Beispiel nicht sofort erkennbar, wie die Entscheidung, sehr früh zum Flughafen aufzubrechen, um das pünktliche Eintreffen zu gewährleisten, einzuordnen ist. Sollte man diese Entscheidung

als fehlertolerierendes oder risikominimierendes Verhalten einstufen? Die Antwort ist ein definitives „Es kommt darauf an...“. Um diese Frage zu beantworten, muss man zunächst genauer definieren, was man unter den Begriffen Fehler und Risiko in dieser Situation versteht. Ist als Fehler das Verpassen des Flugs definiert, so wird zweifelsohne ein frühes Aufbrechen zum Flughafen die Toleranz eines solchen Fehlers nicht erhöhen. Wird allerdings ein unvorhergesehener Stau auf der Autobahn oder gar eine Auto-Panne als möglicher Fehler definiert, so kann man argumentieren, dass die Maximierung der verfügbaren Zeit, den Flughafen rechtzeitig zu erreichen, tatsächlich zu einer Erhöhung der Fehlertoleranz führen kann. Das bedeutet, man kann den definierten Fehler bis zu einem bestimmten Grade tolerieren und trotz einer Verzögerung den Flughafen zeitgerecht erreichen. Wenn man ein System auf seine Fehlertoleranz hin untersuchen möchte, muss man zunächst dessen Ziele und Erwartungswerte genau festlegen.

Während das Konzept der Fehlertoleranz versucht die Effekte eines auftretenden Fehlers zu minimieren, so beschäftigt sich das Konzept der Systemsicherheit damit, die Wahrscheinlichkeit des Auftretens eines Fehlers zu minimieren oder sogar komplett auszuschließen. Die Sicherheit eines Systems wird durch den Designprozess bestimmt, in dem das System so konzipiert wird, dass die Mehrzahl der möglichen Fehlerquellen berücksichtigt werden. Diesbezüglich werden verschiedene Parameter definiert, in deren Schranken das System fehlerfrei operieren muss. Bestimmte Grenzwerte für Temperatur, Feuchtigkeit, Strahlung, Druck, usw. sind Beispiele solcher Parameter. In der Informatik konzentrieren sich solche Parameter allerdings häufig auf Datenbereiche, Datenstrukturen, numerische Grenzwerte und Abweichungen, die für ein Programm oder ein System definiert werden. Die Systemsicherheit wird generell durch verschiedene Tests verifiziert. Die Erstellung dieser Tests, insbesondere im Bereich der Softwareentwicklung, ist nicht trivial, da die Anzahl der Testfälle häufig exponentiell mit der Anzahl der zu berücksichtigenden Parameter ansteigt. Auch in anderen Anwendungsbereichen verlangt die Erstellung adäquater Tests detailliertes Wissen über das System und dessen Anwendungsumgebung.

Als Beispiel eines Systems, von dem extreme Ausfall-Sicherheit gefordert wird, dient der bekannte Flugschreiber (oder *Black Box*), die in jedem Flugzeug installiert ist und verschiedene Systemgrößen der Maschine und Kontrollentscheidungen der Piloten aufzeichnet. Sie dient dazu, im Falle eines Absturzes, die Situation zu rekonstruieren, um die Ursache durch detaillierten Einblick in den Status des Flugzeugs zur Zeit des Absturzes zu erlangen. Natürlich darf diese *Black Box* bei einem Absturz nicht zerstört werden und muss daher so konzipiert werden, dass sie gegen auftretende Kräfte und mögliche Umwelteinflüsse geschützt ist.

Die oben aufgeführten Beispiele sollen helfen, die Konzepte der Fehlertoleranz und der Systemsicherheit zu differenzieren und dem Leser einen ersten Eindruck der Vielfältigkeit dieses Themas zu vermitteln. In der Informatik werden Systeme häufig abstrahiert und durch ein zusammen arbeitendes Netz der verschiedenen Funktionskomponenten dargestellt. Dadurch lässt sich ein solches System einfacher analysieren und auf Fehlerhäufigkeit und Fehlerwahrscheinlichkeit untersuchen.

1.2.1 Fehlertoleranz

Definition 1 (Fehlertoleranz)**.** *Ein System ist* **fehlertolerant***, wenn es trotz des Auftretens unvorhergesehener Fehler weiterhin in der Lage ist seine Funktionen korrekt auszuführen.*

Ein fehlertolerantes System hat somit die Eigenschaft, dass es bei Auftreten von Fehlern graziös degeneriert, d.h. möglicherweise mit verringerter Effizienz oder Genauigkeit weiterarbeitet.

1.2.2 Systemsicherheit

Definition 2 (System Sicherheit)**.** *Ein System gilt als* **sicher***, wenn die Wahrscheinlichkeit auftretender Fehler minimiert ist.*

Das Konzept der Systemsicherheit befasst sich mit der Minimierung des Fehlerrisikos. Natürlich kann man das Auftreten von Fehlern nie völlig ausschließen.

1.2.3 Denkaufgaben

In jedem der folgenden Kontexte sollen je zwei Maßnahmen identifiziert werden, die sowohl die Ausfallsicherheit als auch die Fehlertoleranz erhöhen. Dabei soll genau definiert werden, was als Fehler angesehen werden muss.

1. Fahrzeug (z.B. Motorrad, PKW, usw.)
2. Investition im Aktien- oder Geldmarkt
3. Internet
4. Reiseplanung
5. Industrielle Projektplanung

1.3 Klassifikation möglicher Fehler

In der Umgangssprache verwenden wir Begriffe wie *Fehler, Defekt, Ausfall, und Error.* Während der jeweilige Kontext klar bestimmt was diese Ausdrücke bedeuten, muss man sie für die Fehleranalyse eines Systems genauer definieren. So beschreiben die Begriffe *Fehler, und Defekt* generell das fehlerhafte Verhalten von Systemkomponenten, wobei die Bezeichnung *Error* beschreibt, wie sich ein Fehlverhalten eines Systems manifestiert.
So wird häufig das Verhalten eines logischen Schaltkreises, der unabhängig von den jeweiligen Eingangsgrößen immer den Wert 0 aufweist, als Fehler deklariert. Ein Error tritt auf, wenn dieser Wert dann weiterverarbeitet wird, z.B. in einem Addierer, dessen Ergebnis dann demzufolge fehlerhaft (oder Error-behaftet) ist.
Betrachtet man eine Funktion, bei der dem Programmierer ein *Fehler* bei der Implementierung unterlaufen ist und daher nur positive Funktionswerte berechnet werden. Es präsentiert sich dieser Fehler allerdings nur als *Error*, falls die Funktion mit Parametern aufgerufen wird, die zu einem Ergebnis mit negativen Werten führen.

Generell kann man das Konzept eines Fehlers noch weiter differenzieren:

- **Permanente Fehler** sind Fehler, die bei Auftreten die jeweilige Komponente oder das System permanent außer Betrieb setzen.
- **Transiente Fehler** sind solche Fehler, die dazu führen, dass eine Komponente oder ein System für ein bestimmtes Zeitintervall nicht korrekt arbeitet. Nach diesem Zeitintervall ist der Fehler nicht mehr präsent und das korrekte Systemverhalten ist wiederhergestellt.
- **Sporadische Fehler** sind generell immer präsent, zeigen sich allerdings nur in unregelmäßigen Intervallen als Fehlverhalten des Systems.

Transiente und sporadische Fehler lassen sich häufig sehr schwer diagnostizieren, da ihr Auftreten oft keinem vorhersehbaren Muster folgt. So ist es zum Beispiel schwer einen Programmierfehler zu identifizieren, der nur bei bestimmten Konfigurationen der Eingangswerte oder Parameter auftritt. Hier setzt man spezielle Verfahren der Softwaretechnik ein, die sich mit dem Testen von Software und der Identifikation von Fehlern befassen, die daten- oder konfigurationsabhängig sind.
Viele Fehler können in **unkritische** und **kritische** Fehler klassifiziert werden. So ist beispielsweise ein Fehler, der ein System zu einem kompletten Stillstand bringt als *unkritisch* anzusehen, wenn keine falschen Resultate generiert werden. In diesem Kontext umfassen die *kritischen* Fehler solche, die verfälschte Werte an andere Komponenten weitergeben und somit fehlerhafte Ergebnisse erzeugen. Diese Fehler werden auch häufig als byzantinische Fehler bezeichnet, die wir in einer folgenden Kurseinheit genauer untersuchen werden. Exemplarisch für einen kritischen Fehler betrachtet man ein System, das durch das Austauschen von Messwerten mit anderen Systemen oder Komponenten Entscheidungen koordiniert. Wenn das System allen Komponenten die gleichen (korrekten) Messwerte mitteilt, können alle Teilsysteme unabhängig voneinander Entscheidungen treffen, die mit den bekannten Messwerten korrespondieren. Wenn nun das System aber unterschiedliche (falsche) Messwerte an die Teilsysteme verteilt, so werden diese nicht in der Lage sein, ein konsistentes Ergebnis zu errechnen und somit möglicherweise eine inkonsistente Entscheidung treffen.

1.4 Konzept der Redundanz

Prinzipiell basieren die fehlertolerierenden Eigenschaften eines Systems auf dem Management und Nutzen von **Ressourcen**. Die Verfügbarkeit von mehr Ressourcen als minimal notwendig, um das Funktionieren des Systems zu gewährleisten, wird als **Redundanz** bezeichnet. Dabei umfassen diese Ressourcen nicht nur die notwendigen Hardware- und Softwarekomponenten, sondern auch Zeit und Raum. So kann möglicherweise die räumliche Verteilung oder Ausbreitung von Komponenten zur Fehlertoleranz eines Systems beitragen. Dieser Ansatz wird insbesondere im Gebiet der verteilten Rechnersysteme genutzt, indem ein System so konzipiert wird, dass es keinen zentralen Punkt enthält der durch Auftreten eines Fehlers das gesamte System funktionsunfähig macht.

Die wohl am häufigsten angewandte Form der Redundanz ist die Hardwareredundanz. Dabei werden funktionskritische Elemente eines Systems dupliziert (oder multipliziert), oder es wird eine spezielle Funktion auf unabhängige Komponenten verteilt. Es soll dadurch gewährleistet werden, dass bei Auftreten eines Fehlers die Funktion weiterhin ausgeführt werden kann, obwohl es dabei zu einer Reduzierung der Qualität oder Effizienz kommen kann. Als Beispiel dafür betrachte man eine Zweikreisbremse eines PKW, bei dem die Vorderräder und Hinterräder durch zwei separate Bremskreise gesteuert werden. Sollte in einem Bremskreis ein Fehler auftreten (z.b. durch ein Leck in der Bremsleitung), so ermöglicht der andere Bremskreis noch das Fahrzeug abzubremsen. Die Bremsleistung wird durch das Auftreten eines solchen Fehlers jedoch verringert.

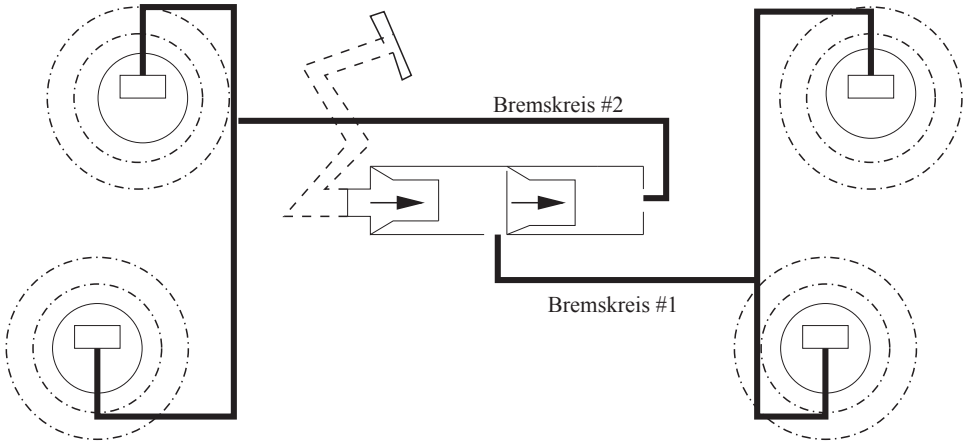

Abbildung 1.1: Zweikreisbremse eines PKW

Weiterhin lässt sich das Konzept der Redundanz in **statische** und **dynamische** Redundanz unterteilen. Während bei statischer Redundanz die redundanten Ressourcen aktiv sind, werden sie bei der dynamischen Redundanz bei Bedarf als *Ersatzkomponenten* aktiviert. Daher wird durch statische Redundanz das Auftreten eines Fehlers maskiert und es ist nicht unmittelbar erkennbar, dass eine Systemkomponente ausgefallen ist. Ein Beispiel für statische Redundanz ist die Verfügbarkeit von Plattenspeichern oder Prozessoren. Das Betriebsystem wird immer versuchen alle verfügbaren Ressourcen zu nutzen, um damit einen Lastausgleich im Rechnersystem zu erreichen. Das Ausfallen dieser Komponenten hat zur Folge, dass diese anfallende zusätzliche Last entweder auf andere Prozessoren verteilt werden muss, oder, im Fall von Speicherfehlern, eine neue Speicherverteilung der Prozesse stattfinden muss.
Ein dynamisches Umschalten auf Ersatzkomponenten erfolgt häufig automatisch, wie zum Beispiel bei einem Netzteil, welches durch einen automatischen Fault-Over im Fehlerfall ein zweites Netzteil aktivieren kann.

1.4.1 Blockdiagramme

Ein System oder ein Prozessablauf wird häufig mittels kanonischer Blockdiagramme abstrahiert. So werden beispielsweise die Komponenten eines Systems und deren Relationen

zueinander als Blockstruktur dargestellt, aus der man die verschiedenen Wege die ein Prozess in dem System nehmen kann, leicht ersehen und analysieren kann. Dabei handelt es sich bei einem solchen Blockdiagramm um einen Graphen G, dessen Knoten die Komponenten des Systems widerspiegeln. Die Kanten von G stellen dar wie die Komponenten voneinander abhängig sind und in welcher Sequenz diese abgearbeitet werden um die Gesamtfunktion des Systems zu realisieren. Während diese Systemdarstellung ihren Ursprung in der Hardware Spezifikation hat, lassen sich aber auch Softwaresysteme durch solche Blockstrukturen analysieren.
Die einfachsten Strukturen bestehen aus seriellen und parallelen Komponenten, aus denen sich die Redundanz des Systems sehr gut darstellen lässt. Ein System, in dem ein Prozess genau vier Komponenten sequentiell durchlaufen muss wird durch das folgende Diagramm dargestellt:

Abbildung 1.2: Diagramm serieller Systemkomponenten

Hier fällt auf, das ein Fehler in irgendeiner der vier Komponenten zu einem nicht funktionierenden System führt. Das heißt, dieses System enthält keine redundante Komponente, die im Fehlerfall die Funktionen der fehlenden Komponente übernehmen kann. Der Ausfall des Gesamtsystems kann aber auch durch einen einfachen logischen Ausdruck definiert werden. Dabei werden die Fehlerstati der Komponenten, $F(U_i)$, durch die logischen Operatoren *UND*, $\wedge$, und *ODER*, $\vee$, miteinander verknüpft.
Es seien $F(U_i)$ ein einfaches Prädikat, dass ausdrücken soll, dass Komponente U_i einen Fehler aufweist. Der logische Zustand des seriellen Gesamtsystems $F(S)$ kann nun durch den folgenden logischen Ausdruck dargestellt werden:

$$F(S) = F(U_1) \vee F(U_2) \vee ... \vee F(U_n)$$

Für das Beispiel mit vier Komponenten in Abbildung 1.2 gilt der Ausdruck:

$$F(S) = F(A) \vee F(B) \vee F(C) \vee F(D)$$

Im Gegensatz zu einem komplett seriellen System kann ein System so konzipiert sein, dass alle Komponenten parallel zueinander angeordnet sind und es somit einen hohen Grad an Redundanz aufweist. Das korrespondierende Blockdiagramm macht dieses deutlich:
Es wird deutlich, das hier nur ein Fehler aller Komponenten zu einem Fehler des Gesamtsystems führen kann. Ähnlich wie bei dem seriellen System, lässt sich das parallele System auch durch einen logischen Ausdruck beschreiben:

$$F(S) = F(U_1) \wedge F(U_2) \wedge ... \wedge F(U_n)$$

Im gegebenen Beispiel (Abbildung 1.3) gilt:

$$F(S) = F(A) \wedge F(B) \wedge F(C) \wedge F(D)$$

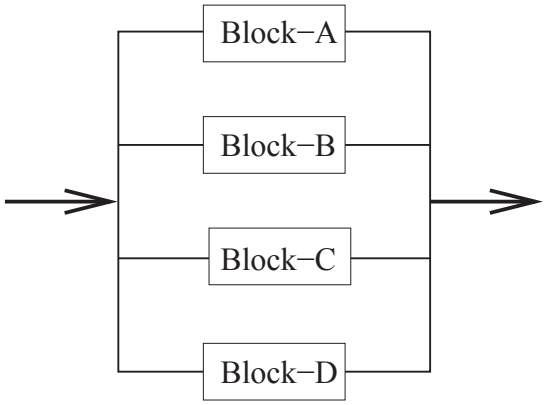

Abbildung 1.3: Diagramm paralleler Systemkomponenten

Wir werden später untersuchen, wie man die Ausfallsicherheit solcher Strukturen berechnen kann wenn die Fehlerwahrscheinlichkeiten der Komponenten bekannt sind. Dazu muss man allerdings Blockstrukturen untersuchen die über die fundamentalen seriellen und parallelen Diagramme hinausgehen. So ist es möglich, serielle und parallele Strukturen zu verbinden um komplexere Systeme darzustellen.

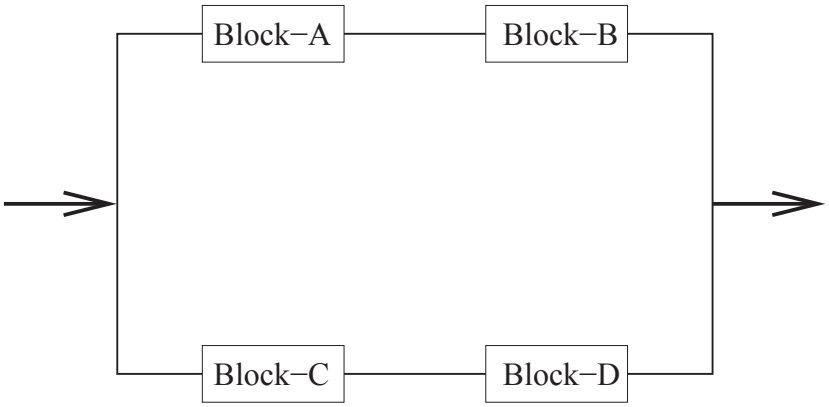

Abbildung 1.4: Serielle-Parallele Systemkomponenten

In dieser Anordnung hängt die Ausfallsicherheit und damit die Fehlertoleranz des Systems davon ab, mit welcher Wahrscheinlichkeit in beiden parallelen Pfaden (also (Block-A, Block-B) oder (Block-C, Block-D)) ein Fehler auftreten kann. Der somit entstehende logische Ausdruck für das Beispiel in Abbildung 1.4, der das Fehlverhalten des Gesamtsystems darstellt, ist somit:

$$F(S) = (F(A) \vee F(B)) \wedge (F(C) \vee F(D))$$

Um diesen Ausdruck zu verallgemeinern definieren wir $\mathcal{P}_{l,r}$ als Menge der parallelen Pfade p_i zwischen zwei Punkten l und r im Blockdiagramm. Zudem definieren wir $\mathcal{S}_{p_i}$ als Menge der seriellen Komponenten, s_j auf einem Pfad p_i. Somit ist der logische Ausdruck, der den Fehlerstatus des Gesamtsystems beschreibt:

$$F(S) = \wedge_{\forall p_i \in \mathcal{P}_{l,r}} [\vee_{\forall s_j \in \mathcal{S}_{p_i}}]$$

Die bis zu diesem Punkt diskutierten Diagramme werden im allgemeinen als Seriell-Parallel oder SP-Diagramme bezeichnet, da sie sich nur aus einfachen seriellen und parallelen Sub-Strukturen zusammensetzen.
Komplexe Systeme lassen sich allerdings nur selten durch einfache SP-Diagramme darstellen und es ist daher notwendig allgemeine, sogenannte Nicht-SP-Strukturen zu untersuchen. Wie später noch genauer gezeigt wird, beeinflusst die Systemstruktur die Methodik mit der die Ausfallsicherheit und die jeweilige Fehlertoleranz eines Systems berechnet und analysiert werden kann.
Im folgenden Beispiel handelt es sich um ein Nicht-SP Blockdiagramm, bei dem der Fehlerstatus des Gesamtsystems durch logische Ausdrücke dargestellt werden kann, die von dem jeweiligen Systempfad abhängig sind.

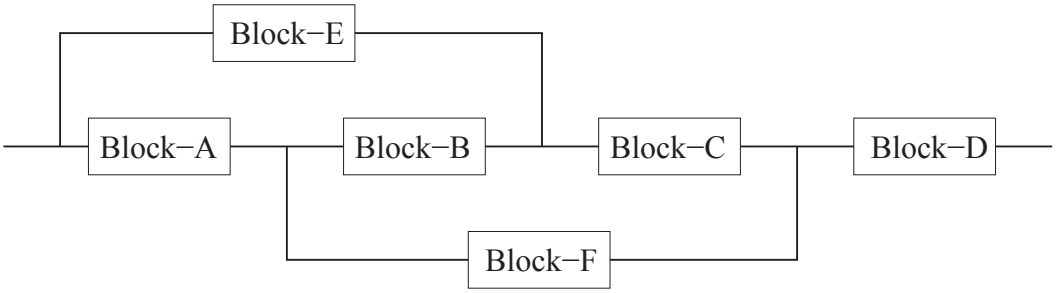

Abbildung 1.5: Nicht-S-P Diagramm

Eine genauere Betrachtung identifiziert *Block D* als eine kritische Systemkomponente, da das Versagen von *Block D* einen Totalausfall des Gesamtsystems hervorruft. Es ist allerdings nicht ganz so offensichtlich, wie sich ein Ausfall der verbleibenden Komponenten auf den Fehlerstaus des Gesamtsystems auswirkt. Um das Fehlerverhalten des Gesamtsystems durch einen logischen Ausdruck zu beschreiben müssen alle möglichen Fehlersituationen berücksichtigt werden, die zu einem Ausfall des Gesamtsystems führen. Man kann dieses Fehlerverhalten einfach durch das Erstellen einer Wertetabelle darstellen, wobei T und F den Fehlerstatus jeder der Komponenten repräsentiert. Wir verwenden hier T für eine fehlerfreie Komponente und F für einen Komponentenfehler. Da *Block D* schon als kritische Komponente identifiziert wurde, wird die folgende Wertetabelle nur den Fehlerstatus der verbleibenden Komponenten darstellen.

Die Ausfallanalyse mittels Wertetabelle ist offensichtlich nur für relativ kleine Subsysteme möglich. Für größere Systeme wird diese Methode schnell unübersichtlich und verlangt somit die Entwicklung rechnergestützter Werkzeuge zur System- und Fehleranalyse.
Generell muss man zwischen *notwendigen* und *hinreichenden* Fehlerbedingungen unterscheiden. So ist das Versagen aller möglichen Pfade zwischen zwei Punkten X_i, X_j im System eine notwendige Fehlerbedingung. Mathematisch wird dies wie folgt beschrieben:

A	B	C	E	F	SYSTEM
T	T	T	T	T	T
T	T	T	T	F	T
T	T	T	F	T	T
T	T	T	F	F	T
T	T	F	T	T	T
T	T	F	T	F	F_2
T	T	F	F	T	T
T	T	F	F	F	F_3
T	F	T	T	T	T
T	F	T	T	F	T
T	F	T	F	T	T
T	F	T	F	F	F_3
T	F	F	T	T	T
T	F	F	T	F	F_3
T	F	F	F	T	T
T	F	F	F	F	F_4
F	T	T	T	T	T
F	T	T	T	F	T
F	T	T	F	T	F_2
F	T	T	F	F	F_3
F	T	F	T	T	F_2
F	T	F	T	F	F_3
F	T	F	F	T	F_3
F	T	F	F	F	F_4
F	F	T	T	T	T
F	F	T	T	F	T
F	F	T	F	T	F_3
F	F	T	F	F	F_4
F	F	F	T	T	F_3
F	F	F	T	F	F_4
F	F	F	F	T	F_4
F	F	F	F	F	F_5

Tabelle 1.1: Tabelle aller möglichen Systemzustände zur Fehleranalyse

$F(S) = 1 \implies \neg\exists(Pfad(X_i, X_j))$, wobei X_i und X_j in unserem System den Eingang und Ausgang zum Gesamtsystem darstellen. Diese Implikation sagt allerdings nichts über die Fehlerbedingungen individueller Komponenten aus, die ausreichen, um alle Pfade zwischen X_i und X_j zu unterbrechen. Die hinreichenden Fehlerbedingungen unseres Beispielsystems sind in der folgenden Tabelle zusammengefasst. Diese Minimalbedingungen (oder hinreichenden Bedingungen) können durch Inspektion der Wertetabelle und des Systemdiagramms hergeleitet werden. Dabei wird eine Bedingung der Form $F(U_1) \wedge F(U_2) \wedge ...F(U_k)$ nur dann als minimal bezeichnet wenn ein Fehler aller k Komponenten für den Ausfall des Gesamtsystems notwendig ist

1 Komponentenfehler	keine (nur Block D im Gesamtsystem)
2 Komponentenfehler	$(A, E), (A, C), (C, F)$
3 Komponentenfehler	(B, E, F)
4 Komponentenfehler	keine

Das bedeutet, dass nur ein Versagen aller möglichen Pfade im System zu einem totalen Systemfehler führt. So ist das gleichzeitige Versagen der Komponenten A und C eine hinreichende minimale Bedingung für das Versagen des Systems. Wie wir später sehen werden, muss die Fehlerwahrscheinlichkeit für jeden dieser Pfade berechnet werden. In Kapitel zwei werden dazu die nötigen theoretischen Grundlagen diskutiert.
Auch ohne die Anwendung theoretischer Grundlagen lassen sich einige Merkmale des Systems in Abbildung 1.5 erkennen. So kann man die Systemkomponente D klar als kritischsten Punkt des Systems identifizieren, da alle möglichen Pfade sie einschließen. Dabei ist Komponente B am unkritischsten, denn nur der Pfad $A - B - C - D$ enthält sie.
Die Ausfallanalyse eines seriell-parallelen (SP) Systems ist generell einfacher als die eines nicht-SP Systems, wie im bearbeiteten Beispiel. Der Grund dafür besteht darin, dass ein SP System hierarchisch zerlegt werden kann und alle seine Komponenten individuell analysiert werden können.

Theorem 1. *Ein Gesamtsystem, bestehend aus Komponenten k_i, kann genau dann durch ein SP Diagramm dargestellt werden, wenn der korrespondierende logische Ausdruck $F(S)$, der den Fehlerstatus des Gesamtsystems ausdrückt, jedes k_i genau einmal enthält.*

Beweis: Der Beweis wird dem Leser als Übungsaufgabe überlassen (siehe Aufgaben am Ende der Kurseinheit).

1.5 Bewertung der Fehlertoleranz

Traditionell hat man Systeme auf ihre Ausfallsicherheit und Verfügbarkeit untersucht, indem man durch Testen die mittleren Zeitintervalle bis zum Auftreten eines Fehlers bestimmt. Dazu wird getestet wie schnell eine fehlerbehaftete Komponente repariert werden kann, da dies die Ausfalldauer direkt beeinflusst. Die Nomenklatur kommt aus dem Englischen und ist in der folgenden Tabelle kurz zusammengefasst:

Englisch	Deutsch	Abkürzung
Reliability	Ausfallsicherheit	$R(t)$
Availability	Verfügbarkeit	$A(t)$
Mean Time to Failure	mittlere ausfallfreie Zeit	MTTF
Mean Time to Repair	mittlere Ausfalldauer	MTTR
Mean Time Between Failures	mittlerer Ausfallabstand	MTBF

Anhand dieser Fehlergrößen kann man nun verschiedene Systemwerte definieren.

Definition 3 (Fehlerrate)**.** *Die zu erwartende Anzahl von auftretenden Systemfehlern, also die Fehlerrate λ, wird wie folgt berechnet:*

$$\lambda = \frac{1}{MTTF}$$

Definition 4 (Reparaturrate)**.** *Die Anzahl der möglichen Reparaturen, die in einer Zeiteinheit durchgeführt werden können wird durch die Reparaturrate μ beschrieben. Da jede Reparatur im Durchschnitt $MTTR$ Zeiteinheiten benötigt, wird die Reparaturrate mit Hilfe der mittleren Ausfalldauer wie folgt berechnet:*

$$\mu = \frac{1}{MTTR}$$

Definition 5 (mittlere Fehlerhäufigkeit)**.** *Die Häufigkeit, mit der Systemfehler auftreten können, werden von den Größen $MTTF$ und $MTTR$ bestimmt. Bei genauerer Betrachtung der Fehlergrößen wird klar, dass die Zeit zwischen zwei Fehlern nicht kleiner als $MTTF + MTTR$ sein kann, denn während der Reparaturzeit ist das System nicht funktionsfähig. Daher wird die mittlere Fehlerhäufigkeit ν wie folgt berechnet:*

$$\nu = \frac{1}{MTTF + MTTR}$$

Mit diesen Definitionen, lässt sich nun die Verfügbarkeit, bzw. die Unverfügbarkeit des Systems ausdrücken. Die *stationäre* Unverfügbarkeit U ist somit

$$U = \frac{MTTR}{MTTF + MTTR} \tag{1.1}$$

Mit $MTBF = MTTF + MTTR$, U kann als

$$U = \frac{MTTR}{MTBF} \tag{1.2}$$

berechnet werden.

Die Unverfügbarkeit ist somit das Verhältnis der durchschnittlichen Reparaturzeit und dem durchschnittlichen Zeitintervall, in dem kein Fehler auftritt. Damit lässt sich die (stationäre) Verfügbarkeit V direkt ausdrücken:

$$V = 1 - U \tag{1.3}$$

Es lässt sich durch algebraische Umformung zeigen das:

$$V = 1 - U = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR} \tag{1.4}$$

Damit beschreibt die Verfügbarkeit das Verhältnis der durchschnittlichen Zeit bis zum Auftreten eines Fehlers und dem durchschnittlichen Zeitintervall, in dem kein Fehler auftritt. Nach einer kurzen Einführung in die Wahrscheinlichkeitsrechnung in der Kurseinheit *Theoretische Grundlagen* wird klar, wie die oben berechneten Fehler und Ausfallraten als Wahrscheinlichkeiten interpretiert werden und somit die Basis für einen stochastischen Ansatz zur Systemanalyse führen.

1.6 Anforderungen an moderne Systeme

Die Anzahl der Operationen und Transaktionen, die moderne Rechner- oder Kommunikationssysteme heute durchführen sollen, verlangen einen hohen Grad an Verfügbarkeit. Schon ein relativ kurzzeitiger Systemausfall kann zu exorbitanten Verlusten führen. Systemingenieure müssen daher ein solches System so konzipieren, dass es den Ansprüchen der jeweiligen Anwendungen gerecht wird. Prinzipiell sind Fehler niemals vermeidbar und somit müssen die einzelnen Systemkomponenten eine Ausfallsicherheit aufweisen, die die geforderte Verfügbarkeit des Gesamtsystems gewährleistet.
Damit lassen sich Ausfallsicherheit und Verfügbarkeit folgendermaßen unterscheiden:

- *Ausfallsicherheit* ist ein (analytisches) vom System festgelegtes Leistungsmerkmal
- *Verfügbarkeit* ist ein vom Benutzer wahrgenommenes Leistungsmerkmal

Es ist somit das Ziel eines fehlertolerierenden Systems die Verfügbarkeit zu maximieren. Eine häufig angewandte Charakterisierung eines Systems ist die **5 x 9** Eigenschaft, die beschreiben soll, dass das System eine Verfügbarkeit von 0.99999 aufweist. Um einen Bezug zu der 5 x 9 Eigenschaft zu entwickeln, betrachte man das folgende Beispiel:

%-Verfügbarkeit V	maximale Ausfallzeit
99%	3 Tage, 15 Stunden, 36 Minuten
99.9%	8 Stunden, 45 Minuten
99.99%	52 Minuten, 36 Sekunden
99.999%	5 Minuten, 15 Sekunden
99.9999%	32 Sekunden

Tabelle 1.2: %-Verfügbarkeit vs. akzeptable Ausfallzeit pro Jahr

Beispiel 1 (Prozentuale Verfügbarkeit)**.** *Ein Jahr hat ca. 525960 Minuten. Damit ergeben sich die folgenden Werte für die Verfügbarkeit und die nicht-planmäßige Ausfallzeit*

Die Werte in Beispiel 1 wurden wie folgt berechnet:

$$1\ Jahr = 525960\ Minuten$$

$$Betriebszeit\ pro\ Jahr = V \times 525960$$

$$U = 525960 - Betriebszeit$$

$$Akzeptable\ Ausfallzeit = (1 - V) * Zeit$$

1.6.1 Beispiele zur Analyse der Verfügbarkeit

Die prozentuale Verfügbarkeit einer Komponente lässt sich leicht anhand des folgenden Beispiels darstellen:

Beispiel 2. *Angenommen die mittlere ausfallfreie Zeit (MTTF) des Netzteils eines Routers ist 18 Jahre. Um einen Fehler zu beheben kann das Netzteil entweder repariert oder ausgetauscht werden. Die Fehlersuche und Reparatur nehmen 24 Stunden in Anspruch ($MTTR_1 = 24h$), während ein Austausch innerhalb von 30 Minuten ausgeführt werden kann ($MTTR_2 = 0.5h$). Wie beeinflusst die Reparatur bzw. der Austausch des Netzteils die Verfügbarkeit?*

Zunächst berechnet man die MTTF in Stunden:

$$MTTF = 18Jahre = 18 \times 365.25 \times 24 = 157788h$$

Die Verfügbarkeit für $MTTF_1$ und $MTTF_2$ ergeben sich dann wie folgt:

$$V_1 = \frac{157788}{157788 + 24} = 0.99984792 \approx 99.985\%$$

$$V_2 = \frac{157788}{157788 + 0.5} = 0.999996831 \approx 99.99968\%$$

1.6.2 Verfügbarkeit eines Systems

Wie schon im Kapitel 1.4.1 beschrieben, lassen sich viele Systeme durch serielle, parallele und seriell-parallele Diagramme beschreiben. Mittels der Verfügbarkeit der einzelnen Komponenten, lässt sich die Verfügbarkeit des Gesamtsystems berechnen. Die Herleitung der jeweiligen Formeln zur Berechnung der Verfügbarkeit werden in Kapitel 2 genauer beschrieben.

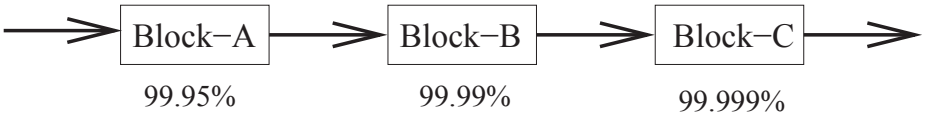

Abbildung 1.6: Diagramm serieller Systemkomponenten

Die Verfügbarkeit eines seriellen Systems ist das Produkt der Verfügbarkeiten der einzelnen Komponenten. Dementsprechend wird die Verfügbarkeit des System im Abbildung 1.6 wie folgt berechnet:

$$V_{sys} = V_A \times V_B \times V_C = 99.95\% \times 99.99\% \times 99.999\% = 99.939\%$$

In einem seriellen System müssen alle Komponenten verfügbar (also funktionsfähig) sein, damit die Funktion des Gesamtsystems gewährleistet ist. Man stellt fest, das V_{sys} immer kleiner ist als die kleinste Verfügbarkeit jeder der Komponenten: $V_{sys} \leq min(V_i) \forall i$

Denkaufgabe: Wie würde man die Unverfügbarkeit U dieses Systems berechnen? (siehe Aufgaben am Ende der Kurseinheit)

Die Produktform zur Berechnung der seriellen Verfügbarkeit muss allerdings modifiziert werden um die Verfügbarkeit eines parallelen Systems zu berechnen.

Für das System in Abbildung 1.7 ergibt sich somit die folgende Formel:

$$V_{sys} = 1 - ((1 - V_A) \times (1 - V_B) \times (1 - V_C)) =$$

$$1 - ((1 - .9995) \times (1 - .9999) \times (1 - .99999)) = .9999999 \approx 99.99999\%$$

In dieser Formel stellen die Terme $(1 - V_i)$ die Unverfügbarkeit der Komponenten dar. Das Produkt der Terme ist daher die Unverfügbarkeit des parallelen Systems, bei dem alle Komponenten ausfallen müssen um einen Fehler des Gesamtsystems zu erzwingen.
Das Berechnen der Verfügbarkeit in einem gemischten seriell-parallel System wird als Übungsaufgabe am Ende dieser Kurseinheit betrachtet.

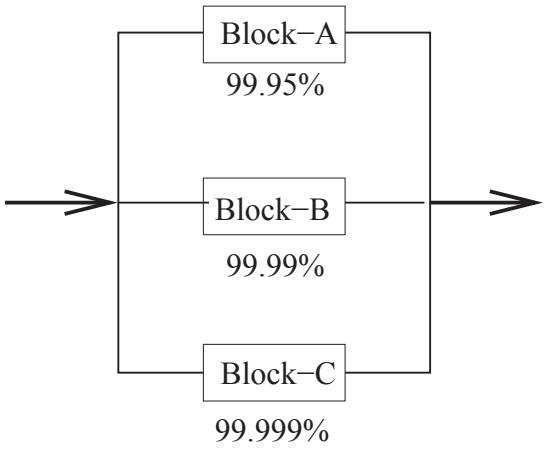

Abbildung 1.7: Diagramm paralleler Systemkomponenten

1.7 Kosten der Fehlertoleranz und Systemsicherheit

Nachdem nun die einführenden Konzepte der Fehlertoleranz vorgestellt wurden, sollte man ein Gefühl dafür entwickeln welche Kosten durch die Fehlertoleranz und Ausfallsicherheit entstehen. Sicherlich sind erhöhte Verfügbarkeit und Ausfallsicherheit mit höheren Entwicklungskosten verbunden, denn eine sorgfältige Systemanalyse im Bezug auf Fehlertoleranz ist recht aufwendig.
Es ist allerdings ungenügend nur die Entwicklungskosten zu betrachten. Die notwendige Redundanz erfordert zusätzliche Komponenten die für das System bereitgestellt werden müssen. Somit schlägt sich die Bereitstellung eines Ersatzrades an einem PKW sicher in seinen Herstellungskosten und Verkaufspreis nieder. Fehlertoleranz und Ausfallsicherheit werden somit wirtschaftstechnische Probleme, die eine Preis-Leistungs- oder Preis-Risiko-Analyse erfordern. Für Systeme deren Ausfall katastrophale Konsequenzen mit sich bringen, sind daher sehr hohe Kosten für die Bereitstellung redundanter Komponenten akzeptabel.
Zusätzlich zu den Kosten der Entwicklung und der Bereitstellung redundanter Komponenten, müssen aber auch Kosten für Zeit, Platz (Raum), und Personal mit in die Kostenrechnung einbezogen werden. So muss zum Beispiel garantiert sein, dass ein Techniker bereitsteht, um das Netzteil, dessen Verfügbarkeit im Beispiel 2 berechnet wurde, austauschen zu können. Nur so kann die erwartete Verfügbarkeit garantiert werden. Da in den meisten Fällen die Fehlertoleranz rund um die Uhr, 365 Tage im Jahr, gewährleistet sein muss, kann der Personalbedarf einen bedeutenden Kostenfaktor darstellen.
Für automatisierte Fehlersysteme werden häufig die redundanten Komponenten in das System integriert um im Fehlerfall automatisch zugeschaltet werden zu können. Das verlangt allerdings eine Vergrößerung der Dimensionen, in denen das System konzipiert ist. Parallele Systeme stellen größere Betriebskapazitäten bereit, verwenden aber nur einen Bruchteil der Kapazität und halten eine Kapazitätsreserve für den Fehlerfall vor. Da dieser Ansatz keine Komponenten ungenutzt lässt, darf dieses System im Normalfall nie 100% ausgelastet sein, um weiterhin auf einen Systemfehler reagieren zu können. Somit muss ungenutzte

Kapazität in die Kostenrechnung mit einbezogen werden.
Es ist häufig schwer die Kosten für Fehlertoleranz und Ausfallsicherheit zu quantifizieren. So wird das zusätzliche Gewicht des Ersatzrads in einem PKW sicher zu einem erhöhtem Kraftstoffverbrauch beitragen. Die extra Zeit, die wir in unserem ersten Beispiel einplanen um rechtzeitig einen Flug zu erreichen, falls ein Unfall oder Stau auf dem Weg zum Flughafen eintritt, erhöht die Anfahrzeit drastisch. Der Aufwand für Zeit und Raum, notwendig um die Ausfallsicherheit und Verfügbarkeit zu erhöhen oder das Fehler Risiko zu senken, lassen sich oft nur sehr schwer quantifizieren.
Die akzeptablen Kosten für die Fehlertoleranz werden nicht nur quantitativ analysiert, sondern unterliegen auch qualitativen Betrachtungen. Fehlerhäufigkeit, Risiko, Daten- Verlust und -Sicherheit werden meist statistisch erfasst und verlangen daher eine Analyse, die auf den Prinzipien der Wahrscheinlichkeitsrechnung basiert. In der folgenden Kurseinheit werden die Axiome der Wahrscheinlichkeitsrechnung wiederholt an Beispielen der Fehlertoleranz erläutert.

1.8 KE-1 Übungsaufgaben

Aufgabe-1:
Für jede der folgenden Fehlerarten sind je zwei Beispiele zu entwickeln:

a. Permanente Fehler;

b. Transiente Fehler;

c. Sporadische Fehler;

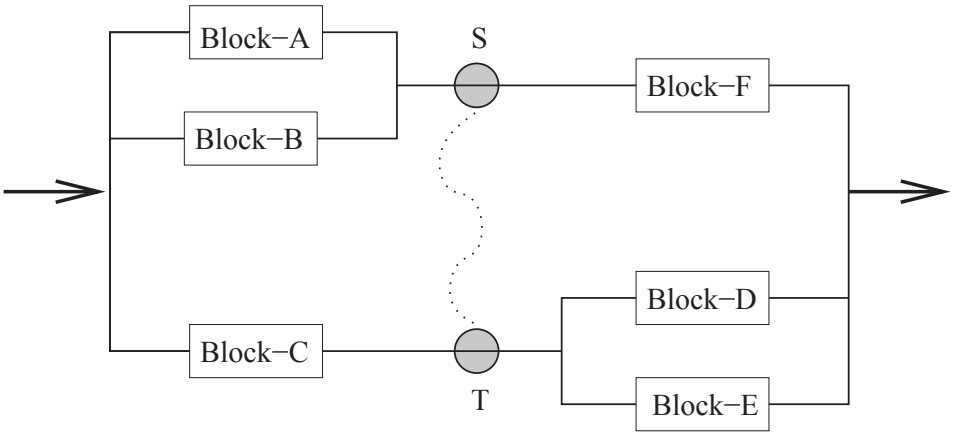

Abbildung 1.8: Blockdiagramm für die folgenden Aufgaben

Aufgabe-2:
Für das Blockdiagramm in Abbildung 1.8 ohne die Verbindung der Punkte S und T ist ein logischer Ausdruck zu entwickeln, der die Fehlerbedingungen des Systems beschreibt.

Wie ändert sich der logische Ausdruck wenn man S und T verbindet? Dabei muss die Durchlaufrichtung berücksichtigt werden:

- Wenn der Systemfluss auf $S \longrightarrow T$ beschränkt ist;
- Wenn der Systemfluss auf $T \longrightarrow S$ beschränkt ist;
- Wenn die Verbindung (S, T) bidirektional durchlaufen werden darf;

Aufgabe-3:
Für das System in Abbildung 1.8 sind die *hinreichenden* Fehlerbedingungen für das Auftreten von $1, 2, \ldots, 6$ Komponentenfehlern zu ermitteln.

Wie ändern sich diese Bedingungen wenn die Verbindung (S, T) berücksichtigt wird? Ändern sich die hinreichenden Bedingungen für einen Systemfehler mit Restriktionen des Systemflusses?

Aufgabe-4:
Es ist formal zu beweisen das:

$$V = 1 - U = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR}$$

Aufgabe-5:
Theorem 1 ist formal zu beweisen.

Aufgabe-6:
Man konstruiere ein Blockdiagramm, welches das fehlertolerante Verhalten eines PKWs mit vier Rädern und einem Ersatzrad darstellt. (Hinweis: Mögliche Kodierung der Elementegruppen)

Aufgabe-7:
Man berechne die Verfügbarkeit des Systems in Abbildung 1.8 wenn die Komponenten $A - F$ eine Verfügbarkeit von je 95% besitzen. Wie verändert sich die Systemverfügbarkeit mit dem Einfügen der (S, T) Verbindung?

Aufgabe-8:
Angenommen das System in Abbildung 1.8 besteht aus Komponenten die innerhalb von 24 Stunden ersetzt oder repariert werden können. Man berechne die Verfügbarkeit des Systems wenn die MTTF für die Komponenten wie folgt vorliegen:

- $Block - A = 12\ Jahre$
- $Block - B = 12\ Jahre$
- $Block - C = 18\ Jahre$
- $Block - D = 15\ Jahre$
- $Block - E = 15\ Jahre$
- $Block - F = 24\ Jahre$