James Forshaw

Netzwerk-protokolle hacken

Sicherheitslücken verstehen, analysieren und schützen

no starch press

dpunkt.verlag

James Forshaw ist ein renommierter Computer-Sicherheits-Experte beim Google-Project Zero und der Entwickler des Netzwerk-Analyse-Tools Canape. Seine Entdeckung von komplexen Designproblemen in Microsoft Windows brachte ihm die »Top-Bug-Prämie« von 100.000 US-Dollar ein und an die Spitze der veröffentlichten Liste des Microsoft Security Response Centers (MSRC). Er wurde eingeladen, seine Ergebnisse auf globalen Sicherheitskonferenzen wie BlackHat, CanSecWest und dem Chaos Computer Congress vorzustellen.

James Forshaw

Netzwerkprotokolle hacken

Sicherheitslücken verstehen, analysieren und schützen

Übersetzung aus dem Amerikanischen
von Peter Klicman

dpunkt.verlag

James Forshaw

Lektorat: Dr. Michael Barabas
Übersetzung: Peter Klicman
Copy-Editing: Ursula Zimpfer, Herrenberg
Satz: Birgit Bäuerlein
Herstellung: Stefanie Weidner
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

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

ISBN:
Print 978-3-86490-569-8
PDF 978-3-96088-473-6
ePub 978-3-96088-474-3
mobi 978-3-96088-475-0

1. Auflage 2018

Wieblinger Weg 17
69123 Heidelberg

5 4 3 2 1 0

Inhaltsübersicht

	Vorwort	**vii**
	Danksagungen	**xi**
	Einführung	**xiii**
	Inhaltsverzeichnis	**xvii**
1	**Netzwerk-Grundlagen**	**1**
2	**Capturing von Anwendungsverkehr**	**13**
3	**Strukturen von Netzwerk-Protokollen**	**41**
4	**Fortgeschrittenes Capturing von Anwendungsverkehr**	**69**
5	**Analyse auf der Datenleitung**	**87**
6	**Reverse Engineering einer Anwendung**	**123**
7	**Sicherheit von Netzwerkprotokollen**	**163**
8	**Implementierung des Netzwerkprotokolls**	**201**
9	**Die Hauptursachen für Sicherheitslücken**	**233**
10	**Sicherheitslücken aufspüren und ausnutzen**	**261**

Anhang

A	**Toolkit für die Netzwerkprotokoll-Analyse**	**311**
	Index	**327**

Vorwort

Als ich James Forshaw zum ersten Mal traf, arbeitete ich in einem Job, den Popular Science 2007 in die Top Ten der miesesten Jobs in der Wissenschaft aufgenommen hatte: als »Sicherheitsknecht bei Microsoft« (»Microsoft Security Grunt«). Das war die recht weit gefasste Bezeichnung, die das Magazin für jeden verwendete, der im Microsoft Security Response Center (MSRC) arbeitete. Was unsere Jobs auf dieser Liste schlimmer als »Walfäkalien-Forscher«, aber doch etwas besser als »Elefanten-Sterilisator« erscheinen ließ, war die enorm hohe Frequenz, mit der Reports über Sicherheitsprobleme in Microsoft-Produkten eingingen. (Die Liste war bei uns in Redmont so bekannt, dass wir uns T-Shirts machen ließen.)

Es geschah hier am MSRC, dass James mit seinem scharfen und kreativen Auge für das Ungewöhnliche und Unbeachtete das erste Mal meine Aufmerksamkeit als Sicherheitsstratege erregte. James war der Autor von einigen der interessantesten Sicherheits-Bug-Reports. Das war durchaus eine Leistung, wenn man bedenkt, dass das MSRC pro Jahr über 200 000 Sicherheits-Bug-Reports von Sicherheitsforschern erhielt. James fand nicht einfach irgendwelche Bugs – er hatte sich das .NET-Framework angesehen und Probleme auf Architekturebene erkannt. Auch wenn diese Bugs auf Architekturebene mit einem einfachen Patch wesentlich schwieriger zu lösen waren, so waren sie doch für Microsoft und seine Kunden sehr wertvoll.

Kommen wir gleich zu Microsofts erstem Bug-Prämien-Programm, das ich im Unternehmen im Juni 2013 einführte. Bei diesem ersten Paket von Bug-Prämien gab es drei Programme. Diese Programme versprachen Sicherheitsforschern wie James Geld für die Meldung schwerwiegender Bugs an Microsoft. Ich wusste, dass qualitativ hochwertige Sicherheits-Bugs eingereicht werden mussten, um die Effizienz des Programms nachzuweisen.

Als wir es auflegten, gab es keine Garantie, dass die Bug-Sucher zu uns kommen würden. Wir wussten, dass wir um einige der höchstqualifizierten Bug-Jäger auf der Welt konkurrierten. Es gab zahlreiche andere Geldprämienprogramme und nicht alle Marktplätze für Bugs dienten der Verteidigung. Nationalstaaten und Kriminelle betrieben etablierte, auf Angriffe ausgerichtete Märkte für Bugs und Exploits, und Microsoft war auf diejenigen Bug-Sucher angewiesen, die kos-

tenlos über 200 000 Bug-Reports pro Jahr einreichten. Die Prämien sollten die Aufmerksamkeit dieser netten, altruistischen Bug-Jäger auf die Probleme richten, bei deren Lösung Microsoft die meiste Hilfe benötigte.

Also rief ich James und eine Handvoll anderer Leute an, weil ich darauf zählte, dass sie die fehlerbehafteten Produkte aufspüren würden. Bei diesen ersten Microsoft Bug-Prämien wünschten wir Sicherheitsknechte am MSRC uns das Aufdecken von Sicherheitslücken für den Internet Explorer (IE) 11 Beta, und wir wollten etwas, wofür ein Softwarehersteller noch nie eine Bug-Prämie ausgelobt hatte: Wir wollten Informationen über neue Exploit-Techniken erfahren. Diese Fangprämie war als »Mitigation Bypass Bounty« bekannt und damals 100 000 Dollar wert.

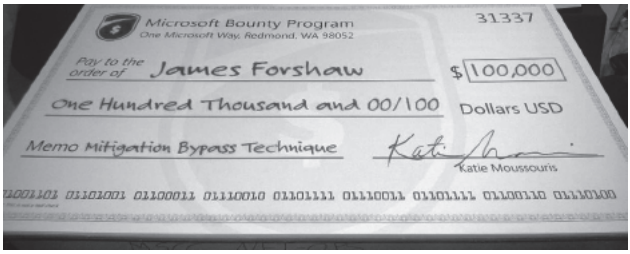
Ich erinnere mich daran, wie ich mit James in London bei einem Bier zusammensaß und ihn zu überzeugen versuchte, nach IE-Bugs zu suchen. Er erklärte mir, dass er sich noch nie mit der Sicherheit von Browsern beschäftigt hatte und dass ich nicht zu viel von ihm erwarten sollte.

Dennoch reichte James vier verschiedene Sandbox-Escapes für IE 11 Beta ein. *Vier*!

Diese Sandbox-Escapes lagen alle in Bereichen des IE-Codes, die unsere internen Teams und externen, privaten Pentester übersehen hatten. Mithilfe von Sandbox-Escapes können andere Bugs zuverlässiger ausgenutzt werden. James erhielt vom IE-Team selbst Prämien für alle vier Bugs sowie einen zusätzlichen Bonus von 5 000 Dollar aus meinem Prämienbudget. Zurückblickend hätte ich ihm wohl 50 000 Dollar geben sollen. Wow, nicht schlecht für jemanden, der sich vorher noch nie mit der Sicherheit von Webbrowsern beschäftigt hatte.

Nur einige Monate später stand ich vor einer Microsoft-Cafeteria und rief völlig atemlos James an einem stürmischen Herbsttag an, um ihm zu berichten, dass er gerade Geschichte geschrieben hatte. Ich konnte es gar nicht abwarten, ihm zu erzählen, dass sein Beitrag für eines der anderen Bug-Prämien-Programme von Microsoft – das Mitigation Bypass Bounty für 100 000 Dollar – akzeptiert worden war. James Forshaw hatte eine neuartige Möglichkeit gefunden, alle Plattform-Verteidigungslinien zu umgehen, indem er architektonische Sicherheitslücken des neuesten Betriebssystems ausnutzte. Er gewann damit die allererste 100 000-Dollar-Prämie von Microsoft.

Soweit ich mich erinnere, stellte er sich bei diesem Telefongespräch vor, wie ich ihm bei der Microsoft-internen BlueHat-Konferenz auf der Bühne einen witzigen, riesigen Scheck überreiche. Ich schickte nach dem Telefonat eine Notiz an die Marketingabteilung und im Handumdrehen wurde »James und der Riesenscheck« für immer Teil der Microsoft- und Internet-Geschichte.



Ich bin mir sicher, dass der Leser aus den folgenden Seiten einen Teil von James' unvergleichlicher Brillanz mitnehmen kann – die gleiche Brillanz, die ich vor vielen Jahren in einem Bug-Report (oder vier) sah. Es gibt nur wenige Sicherheitsforscher, die Bugs in einer fortgeschrittenen Technologie finden können, und noch weniger, die sie durchgängig in mehr als einer finden. Und dann gibt es Menschen wie James Forshaw, die sich mit der Präzision eines Chirurgen auf tieferliegende architektonische Aspekte konzentrieren. Ich hoffe, dass die Leser dieses Buches (und aller Bücher, die James noch schreiben wird) es als Praxisleitfaden nutzen, um die gleiche Brillanz und Kreativität in ihrer eigenen Arbeit zu erzielen.

In einem Bug-Prämien-Meeting bei Microsoft, wo die Mitglieder des IE-Teams den Kopf schüttelten und sich fragten, wie sie einige von James gemeldeten Bugs hatten übersehen können, erklärte ich einfach: »James kann die Frau in Rot sehen sowie den Code, der sie in der Matrix gerendert hat.« Alle am Tisch akzeptierten diese Erklärung für den in James arbeitenden Geist. Er kann so ziemlich jeden Löffel verbiegen und wenn Sie seine Arbeit studieren (und einen offenen Geist haben), können Sie das vielleicht auch.

Für alle Bug-Jäger auf der Welt liegt hier die Messlatte und sie ist hoch. Und all den ungenannten Sicherheitsknechten auf der ganzen Welt wünsche ich, dass ihre Bug-Reports ebenso interessant und wertvoll sind wie die von James Forshaw.

Katie Moussouris
Gründerin und CEO, Luta Security
Oktober 2017

Danksagungen

Vielen Dank, dass Sie dieses Buch lesen. Ich hoffe, Sie finden es aufschlussreich und von praktischem Nutzen. Viele verschiedene Menschen haben dazu beigetragen, worüber ich dankbar bin.

Ich muss mit einem Dank an meine Frau Huayi beginnen, die dafür sorgte, dass ich beim Schreiben blieb, auch wenn ich es nicht wollte. Dank ihrer Unterstützung ist es in nur vier Jahren fertig geworden. Ohne sie hätte ich es vielleicht in zwei Jahren schaffen können, doch es hätte nicht so viel Spaß gemacht.

Natürlich wäre ich ohne meine wunderbaren Eltern heute nicht hier. Ihre Liebe und Unterstützung hat dazu geführt, dass ich ein weithin anerkannter Sicherheitsforscher und Autor bin. Als ich jung war, kauften sie für die Familie einen Computer – einen Atari 400 –, und sie waren es, die mein Interesse für Computer und die Softwareentwicklung geweckt haben. Ich kann ihnen nicht genug dafür danken, mir all diese Möglichkeiten gegeben zu haben.

Der große Kontrapunkt zu meinem Leben als Computer-Nerd war mein ältester Freund Sam Shearon. Er war immer der Selbstbewusstere und Kontaktfreudigere und darüber hinaus ein begnadeter Künstler, der mir eine andere Seite des Lebens zeigte.

Im Verlauf meiner Karriere gab es viele Kollegen und Freunde, die wesentlich zu meinen Erfolgen beigetragen haben. Unter ihnen muss ich Richard Neal hervorheben, einen guten Freund und manchmal Vorgesetzten, der mir die Gelegenheit gab, mich für Computersicherheit zu interessieren. Die erforderlichen Fähigkeiten kamen meiner Mentalität entgegen.

Ich darf auch Mike Jordon nicht vergessen, der mich überzeugte, für Context Information Security in Großbritannien zu arbeiten. Zusammen mit den Inhabern Alex Church und Mark Raeburn gab er mir die Zeit, eindrucksvolle Sicherheitsforschung zu betreiben, mein Wissen um die Analyse von Netzwerkprotokollen zu erweitern und Werkzeuge wie Canape zu entwickeln. Diese Erfahrung, reale und üblicherweise vollständig maßgeschneiderte Netzwerkprotokolle anzugreifen, bildet den größten Teil des Inhalts dieses Buches.

Ich muss Katie Moussouris danken, die mich überredet hat, am Microsoft Mitigation Bypass Bounty teilzunehmen, was mein Ansehen in der Welt der

Informationssicherheit deutlich steigerte und mir für meine Mühen auch einen Scheck über 100 000 Dollar einbrachte.

Mein gestiegenes Ansehen war auch von Nutzen, als das Team für Google Project Zero – eine Gruppe weltweit führender Sicherheitsforscher mit dem Ziel, die Plattformen, von denen wir alle abhängig sind, sicherer zu machen – aufgebaut wurde. Will Harris erwähnte meinen Namen gegenüber Chris Evans, dem aktuellen Chef des Teams, der mich zu einem Interview einlud, und plötzlich war ich ein Googler. Mitglied eines solch exzellenten Teams zu sein macht mich stolz.

Zum Schluss muss ich Bill, Laurel und Liz von No Starch Press danken, die die Geduld hatten, mich dieses Buch fertig schreiben zu lassen, und mir Ratschläge gaben, wie ich es angehen sollte. Ich hoffe, dass sie – und die Leser – mit dem Ergebnis zufrieden sind.

Einführung

Bei ihrer Einführung stand die Technik, die es Geräten erlaubte, sich zu einem Netzwerk zu verbinden, nur großen Unternehmen und Regierungen zur Verfügung. Heutzutage tragen die meisten Menschen ein voll vernetztes Gerät mit sich herum und mit dem Aufkommen des Internets der Dinge (Internet of Things, IoT) können Sie Ihren Kühlschrank und die heimische Alarmanlage an diese vernetzte Welt anbinden. Die Sicherheit dieser vernetzten Geräte wird daher immer wichtiger. Möglicherweise kümmert es Sie nicht sonderlich, wenn jemand weiß, welchen Joghurt Sie kaufen, doch wenn Ihr Smartphone über das gleiche Netzwerk wie Ihr Kühlschrank kompromittiert wird, könnten all Ihre persönlichen und finanziellen Daten in die Hände böser Hacker gelangen.

Dieses Buch heißt *Netzwerkprotokolle hacken*, weil Sie sich in die Gedankenwelt eines Angreifers hineinversetzen müssen, um Sicherheitslücken bei einem vernetzten Gerät aufzuspüren. Netzwerkprotokolle kommunizieren mit anderen Geräten über ein (öffentliches) Netzwerk. Sie durchlaufen häufig nicht die Prüfungen wie die anderen Komponenten des Gerätes und sind daher ein nahe liegendes Angriffsziel.

Warum sollten Sie dieses Buch lesen?

Viele Bücher behandeln das Erfassen (Capturing) von Netzwerkverkehr für Diagnosezwecke und eine grundlegende Netzwerkanalyse, kümmern sich aber nicht um die Sicherheitsaspekte der abgefangenen Protokolle. Im Gegensatz dazu konzentriert sich dieses Buch darauf, Protokolle auf ihre Sicherheitslücken hin zu analysieren.

Dieses Buch richtet sich an alle, die Netzwerkprotokolle analysieren und angreifen wollen, aber nicht wissen, wo sie anfangen sollen. Die Kapitel vermitteln Ihnen Techniken zum Erfassen von Netzwerkverkehr, zur Analyse der Protokolle und zur Aufdeckung und dem Exploit von Sicherheitslücken. Das Buch bietet Hintergrundinformationen zur Vernetzung und zur Sicherheit von Netzwerken, aber auch praktische Beispiel für zu analysierende Protokolle.

Ob Sie nun Netzwerkprotokolle angreifen wollen, um Sicherheitslücken an den Hersteller einer Anwendung zu melden, oder ob Sie nur wissen wollen, wie Ihr neuestes IoT-Gerät kommuniziert, Sie werden verschiedene interessante Themen entdecken.

Was finden Sie in diesem Buch?

Dieses Buch umfasst eine Mischung aus theoretischen und praktischen Kapiteln. Für die praktischen Kapitel habe ich eine Netzwerkbibliothek namens Canape Core entwickelt und zur Verfügung gestellt, mit der Sie eigene Tools zur Protokollanalyse und für Exploits schreiben können. Ich stelle auch eine beispielhafte Netzwerkanwendung namens *SuperFunkyChat* bereit, die ein benutzerdefiniertes Chat-Protokoll implementiert. Während Sie der Diskussion in den Kapiteln folgen, können Sie die Beispielanwendung nutzen, um die Protokollanalyse zu erlernen und die Beispielprotokolle anzugreifen. Hier eine kurze Beschreibung der jeweiligen Kapitel:

- **Kapitel 1: Netzwerk-Grundlagen**
  Dieses Kapitel erläutert die Grundlagen der Vernetzung von Computern, wobei es sich auf TCP/IP konzentriert, das die Basis anwendungsspezifischer Netzwerkprotokolle bildet. Die nachfolgenden Kapitel gehen davon aus, dass Sie die Grundlagen der Vernetzung beherrschen. Dieses Kapitel stellt auch den Ansatz vor, den ich zur Modellierung von Anwendungsprotokollen verwende. Dieses Modell teilt das Anwendungsprotokoll in flexible Schichten auf und abstrahiert komplexe technische Details. Auf diese Weise können wir uns auf bestimmte Teile des zu analysierenden Protokolls konzentrieren.
- **Kapitel 2: Capturing von Anwendungsverkehr**
  Dieses Kapitel führt in die Konzepte des passiven und aktiven Capturings von Netzwerkverkehr ein. Es ist das erste Kapitel, das die Canape-Core-Bibliotheken für praktische Aufgaben nutzt.
- **Kapitel 3: Strukturen von Netzwerk-Protokollen**
  Dieses Kapitel erläutert interne Strukturen, die bei Netzwerkprotokollen üblich sind, etwa die Repräsentation von Zahlen oder lesbarem Text. Bei der Analyse von Netzwerkverkehr können Sie dieses Wissen nutzen, um gängige Strukturen schnell zu identifizieren, und so die Analyse beschleunigen.
- **Kapitel 4: Fortgeschrittenes Capturing von Anwendungsverkehr**
  Dieses Kapitel untersucht eine Reihe fortgeschrittener Capturing-Techniken, die die Beispiele aus Kapitel 2 ergänzen. Zu diesen Techniken gehören etwa die »Network Address Translation« zur Umleitung von Verkehr und das Spoofing von ARP.

- **Kapitel 5: Analyse auf der Datenleitung**
  Dieses Kapitel stellt Methoden zur Analyse des aufgezeichneten Netzwerkverkehrs vor und nutzt dabei die in Kapitel 2 erläuterten passiven und aktiven Techniken. In diesem Kapitel lassen wir die *SuperFunkyChat*-Anwendung erstmals Beispielverkehr erzeugen.
- **Kapitel 6: Reverse Engineering einer Anwendung**
  In diesem Kapitel werden Techniken zum Reverse Engineering von Netzwerkprogrammen erläutert. Reverse Engineering erlaubt die Analyse eines Protokolls, ohne dass Sie dazu Beispielverkehr benötigen. Diese Methoden helfen auch dabei, die verwendete Verschlüsselungs- oder Verschleierungstechnik zu identifizieren, sodass sich der aufgezeichnete Verkehr besser analysieren lässt.
- **Kapitel 7: Sicherheit von Netzwerkprotokollen**
  Dieses Kapitel versorgt Sie mit Hintergrundinformationen zu Techniken und kryptografischen Algorithmen, die zur Absicherung von Netzwerkprotokollen verwendet werden. Der Schutz der über öffentliche Netzwerke laufenden Daten vor Enthüllung oder Veränderung ist für die Sicherheit des Netzwerkprotokolls von höchster Bedeutung.
- **Kapitel 8: Implementierung des Netzwerkprotokolls**
  Dieses Kapitel erläutert Techniken zur Implementierung des Anwendungsprotokolls in selbst entwickeltem Code. Auf diese Weise können Sie das Verhalten des Protokolls testen und Sicherheitslücken aufspüren.
- **Kapitel 9: Die Hauptursachen für Sicherheitslücken**
  Dieses Kapitel zeigt gängige Sicherheitslücken auf, denen Sie bei einem Netzwerkprotokoll begegnen werden. Wenn Sie die Hauptursachen für Sicherheitslücken kennen, können Sie diese während der Analyse einfacher identifizieren.
- **Kapitel 10: Sicherheitslücken aufspüren und ausnutzen**
  Dieses Kapitel beschreibt den Prozess der Aufspürens von Sicherheitslücken anhand der Hauptursachen aus Kapitel 9 und demonstriert eine Reihe von Möglichkeiten, wie Sie das ausnutzen können. Dazu entwickeln wir eigenen Shell-Code und umgehen möglicherweise getroffene Gegenmaßnahmen durch »Return-Oriented Programming«.
- **Anhang A: Toolkit für die Netzwerkprotokoll-Analyse**
  In diesem Anhang finden Sie die Beschreibung einiger der Tools, die ich zur Protokollanalyse häufig einsetze. Viele dieser Tools werden auch im Text angesprochen.

Wie Sie dieses Buch nutzen

Wenn Sie Ihr Grundlagenwissen in Sachen Vernetzung auffrischen wollen, lesen Sie zuerst Kapitel 1. Wenn Sie mit den Grundlagen vertraut sind, können Sie mit den Kapiteln 2 und 3 weitermachen sowie in Kapitel 5 praktische Erfahrungen mit dem Aufzeichnen von Netzwerkverkehr und dem Analyseprozess sammeln.

Mit dem Wissen um die Grundlagen des Erfassens und der Analyse können Sie mit den Kapiteln 7 bis 10 weitermachen. Darin finden Sie praxisorientierte Hinweise, wie man Sicherheitslücken dieser Protokolle aufspürt und ausnutzt. Die Kapitel 4 und 6 enthalten weiterführende Informationen zu zusätzlichen Capturing-Techniken und dem Reverse Engineering von Anwendungen. Wenn Sie wollen, können Sie diese lesen, nachdem Sie die anderen Kapitel durchgelesen haben.

Für die praktischen Beispiele müssen Sie .NET Core (*https://www.microsoft.com/net/core/*) installieren. Das ist die plattformübergreifende Version der .NET-Runtime von Microsoft, die unter Windows, Linux und macOS läuft. Sie können Versionen von Canape Core über *https://github.com/tyranid/CANAPE.Core/releases/* und *SuperFunkyChat* über *https://github.com/tyranid/ExampleChatApplication/releases/* herunterladen. Beide nutzen .NET Core als Runtime. Links zu diesen Websites finden Sie in den Ressourcen zu diesem Buch auf *https://www.dpunkt.de/netze_hacken*.

Um die Canape-Core-Beispielskripten auszuführen, müssen Sie *CANAPE.Cli* nutzen, das im Releasepaket enthalten ist, das Sie aus dem Github-Repository zu Canape Core heruntergeladen haben. Führen Sie das Skript mit der folgenden Kommandozeile aus und ersetzen Sie dabei `script.csx` durch den Namen des Skripts, das Sie ausführen wollen.

```
dotnet exec CANAPE.Cli.dll script.csx
```

Alle Beispiel-Listings aus den praktischen Kapiteln sowie die Paket-Captures (erfassten Pakete) stehen auf der Webseite zu diesem Buch unter *https://www.dpunkt.de/netze_hacken* zur Verfügung. Bevor Sie anfangen, sollten Sie sich diese Beispiele herunterladen, damit Sie den praktischen Kapiteln folgen können, ohne eine große Menge Quellcode von Hand eingeben zu müssen.

Kontakt

Ich bin immer an positivem wie negativem Feedback zu meiner Arbeit interessiert und dieses Buch bildet da keine Ausnahme. Sie erreichen mich per E-Mail unter *attacking.network.protocols@gmail.com*.

Sie können mir auch auf Twitter unter *@tiraniddo* folgen oder meinen Blog unter *https://tyranidslair.blogspot.com/* abonnieren, wo ich über meine neuesten Forschungen zur IT-Sicherheit schreibe.

Inhaltsverzeichnis

1 Netzwerk-Grundlagen 1

1.1 Netzwerkarchitekturen und -protokolle 1
1.2 Die Internet-Protokoll-Suite 3
1.3 Datenkapselung 5
1.3.1 Header, Footer und Adressen 5
1.3.2 Datenübertragung 7
1.4 Netzwerk-Routing 8
1.5 Mein Modell für die Analyse von Netzwerkprotokollen 9
1.6 Am Ende dieses Kapitels 12

2 Capturing von Anwendungsverkehr 13

2.1 Passives Capturing von Netzwerkverkehr 13
2.2 Eine kurze Einführung in Wireshark 14
2.3 Alternative passive Capturing-Techniken 16
2.3.1 Tracing von Systemaufrufen 17
2.3.2 Das strace-Utility unter Linux 18
2.3.3 Netzwerkverbindungen mit DTrace verfolgen 19
2.3.4 Process Monitor unter Windows 21
2.4 Vor- und Nachteile passiven Capturings 22
2.5 Aktives Capturing von Netzwerkverkehr 23
2.6 Netzwerk-Proxys 23
2.6.1 Port-Forwarding-Proxy 24
2.6.2 SOCKS-Proxy 28
2.6.3 HTTP-Proxys 33
2.6.4 Forwarding eines HTTP-Proxys 33
2.6.5 HTTP-Reverse-Proxy 37
2.7 Am Ende dieses Kapitels 40

3 Strukturen von Netzwerk-Protokollen 41

3.1 Binäre Protokollstrukturen 42
3.1.1 Numerische Daten 42
3.1.2 Boolesche Werte 45
3.1.3 Bit-Flags 46
3.1.4 Binäre Bytereihenfolge (Endianness) 46
3.1.5 Text und menschenlesbare Daten 47
3.1.6 Binärdaten variabler Länge 52
3.2 Datum und Uhrzeit 55
3.2.1 POSIX/Unix-Zeit 55
3.2.2 Windows FILETIME 55
3.3 TLV-Muster 56
3.4 Multiplexing und Fragmentierung 57
3.5 Netzwerk-Adressinformationen 58
3.6 Strukturierte Binärformate 59
3.7 Strukturen textbasierter Protokolle 60
3.7.1 Numerische Daten 61
3.7.2 Boolesche Werte in Textform 61
3.7.3 Datum und Uhrzeit 61
3.7.4 Daten variabler Länge 62
3.7.5 Formate für strukturierten Text 63
3.8 Codierung binärer Daten 65
3.8.1 Hex-Codierung 66
3.8.2 Base64 66
3.9 Am Ende dieses Kapitels 68

4 Fortgeschrittenes Capturing von Anwendungsverkehr 69

4.1 Rerouting von Verkehr 69
4.1.1 Traceroute nutzen 70
4.1.2 Routing-Tabellen 71
4.2 Konfiguration eines Routers 72
4.2.1 Routing unter Windows aktivieren 73
4.2.2 Routing unter *nix aktivieren 73
4.3 Network Address Translation 74
4.3.1 SNAT aktivieren 74
4.3.2 SNAT unter Linux konfigurieren 75
4.3.3 DNAT aktivieren 76

4.4 Verkehr an ein Gateway weiterleiten 78
4.4.1 DHCP-Spoofing 78
4.4.2 ARP-Poisoning 81
4.5 Am Ende dieses Kapitels 85

5 Analyse auf der Datenleitung 87

5.1 Die Verkehr produzierende Anwendung: SuperFunkyChat 87
5.1.1 Den Server starten 88
5.1.2 Clients starten 88
5.1.3 Kommunikation zwischen Clients 89
5.2 Ein Crashkurs zur Analyse mit Wireshark 90
5.2.1 Netzwerkverkehr generieren und Pakete erfassen 91
5.2.2 Grundlegende Analyse 93
5.2.3 Inhalte einer TCP-Session lesen 94
5.3 Die Paketstruktur mit Hex Dump identifizieren 95
5.3.1 Einzelne Pakete betrachten 96
5.3.2 Die Protokollstruktur ermitteln 97
5.3.3 Unsere Annahmen überprüfen 99
5.3.4 Das Protokoll mit Python sezieren 100
5.4 Einen Wireshark-Dissector in Lua entwickeln 106
5.4.1 Den Dissector entwickeln 109
5.4.2 Sezieren mit Lua 110
5.4.3 Parsen eines Nachrichtenpakets 111
5.5 Einen Proxy zur aktiven Verkehrsanalyse nutzen 114
5.5.1 Den Proxy einrichten 115
5.5.2 Protokollanalyse mittels Proxy 117
5.5.3 Grundlegendes Parsen von Protokollen hinzufügen 119
5.5.4 Das Protokollverhalten ändern 120
5.6 Am Ende dieses Kapitels 122

6 Reverse Engineering einer Anwendung 123

6.1 Compiler, Interpreter und Assembler 124
6.1.1 Interpretierte Sprachen 124
6.1.2 Kompilierte Sprachen 125
6.1.3 Statisches und dynamisches Linking 125
6.2 Die x86-Architektur 126
6.2.1 Instruction Set Architecture 127
6.2.2 CPU-Register 128
6.2.3 Ablaufsteuerung 131

6.3 Betriebssystem-Grundlagen 132
6.3.1 Dateiformate für Executables 132
6.3.2 Abschnitte 133
6.3.3 Prozesse und Threads 133
6.3.4 Netzwerkschnittstelle des Betriebssystems 134
6.3.5 Application Binary Interface 137
6.4 Statisches Reverse Engineering 138
6.4.1 Kurzanleitung für die Nutzung der IDA Pro Free Edition . . 139
6.4.2 Stackvariablen und Argumente analysieren 143
6.4.3 Schlüsselfunktionalitäten identifizieren 143
6.5 Dynamisches Reverse Engineering 150
6.5.1 Breakpunkte setzen 151
6.5.2 Debugger-Fenster 151
6.5.3 Wo setzt man Breakpunkte? 153
6.6 Reverse Engineering von Managed Code 153
6.6.1 .NET-Anwendungen 154
6.6.2 ILSpy nutzen 155
6.6.3 Java-Anwendungen 158
6.6.4 Mit Verschleierungstaktiken umgehen 160
6.7 Reverse-Engineering-Ressourcen 161
6.8 Am Ende dieses Kapitels 161

7 Sicherheit von Netzwerkprotokollen 163

7.1 Verschlüsselungsalgorithmen 164
7.1.1 Substitutionschiffre 165
7.1.2 XOR-Verschlüsselung 166
7.2 Zufallszahlengeneratoren 167
7.3 Symmetrische Verschlüsselung 168
7.3.1 Blockchiffre 168
7.3.2 Blockchiffre-Modi 171
7.3.3 Blockchiffre-Padding 174
7.3.4 Padding Oracle Attack 176
7.3.5 Stromchiffre 178
7.4 Asymmetrische Verschlüsselung 179
7.4.1 RSA-Algorithmus 180
7.4.2 RSA-Padding 182
7.4.3 Schlüsselaustausch nach Diffie-Hellman 182
7.5 Signaturalgorithmen 184
7.5.1 Kryptografische Hash-Algorithmen 185
7.5.2 Asymmetrische Signaturalgorithmen 186
7.5.3 Message Authentication Codes 187

7.6 Public-Key-Infrastruktur 190
7.6.1 X.509-Zertifikate 190
7.6.2 Verifikation einer Zertifikatskette 192
7.7 Fallbeispiel: Transport Layer Security 193
7.7.1 Der TLS-Handshake 194
7.7.2 Initiale Aushandlungen 195
7.7.3 Endpunkt-Authentifizierung 195
7.7.4 Die Verschlüsselung aufbauen 197
7.7.5 Sicherheitsanforderungen erfüllen 198
7.8 Am Ende dieses Kapitels 200

8 Implementierung des Netzwerkprotokolls 201

8.1 Replay von erfasstem Netzwerkverkehr 201
8.1.1 Verkehr mit Netcat erfassen 202
8.1.2 Replay von UDP-Verkehr mittels Python 204
8.1.3 Unseren Analyse-Proxy wiederverwenden 206
8.2 Ausführbaren Code wiederverwenden 211
8.2.1 Code in .NET-Anwendungen wiederverwenden 212
8.2.2 Code in Java-Anwendungen wiederverwenden 217
8.2.3 Unmanaged Executables 219
8.3 Verschlüsselung und der Umgang mit TLS 224
8.3.1 Die verwendete Verschlüsselung ermitteln 225
8.3.2 TLS-Verkehr entschlüsseln 226
8.4 Am Ende dieses Kapitels 232

9 Die Hauptursachen für Sicherheitslücken 233

9.1 Vulnerabilitätsklassen 234
9.1.1 Remote Code Execution 234
9.1.2 Denial-of-Service 234
9.1.3 Offenlegung von Informationen 235
9.1.4 Authentifizierung umgehen 235
9.1.5 Autorisierung umgehen 235
9.2 Verfälschung des Speichers 236
9.2.1 Speichersichere und speicherunsichere Programmiersprachen 236
9.2.2 Pufferüberlauf 237
9.2.3 Out-of-Bounds-Indexierung 242
9.2.4 Datenexpansion 243
9.2.5 Fehler bei der dynamischen Speicherallozierung 244
9.3 Voreingestellte oder festcodierte Anmeldedaten 244
9.4 Offenlegung von Benutzernamen 245

9.5 Fehlerhafter Zugriff auf Ressourcen 247
9.5.1 Kanonisierung 247
9.5.2 Fehlermeldungen mit zu viel Information 249
9.6 Speicherüberlastung 250
9.7 Massenspeicherüberlastung 251
9.8 CPU-Überlastung 252
9.8.1 Algorithmische Komplexität 252
9.8.2 Konfigurierbare Kryptografie 254
9.9 Formatstrings 255
9.10 Befehlsinjektion 256
9.11 SQL-Injektion 257
9.12 Zeichenersetzung bei Textcodierung 258
9.13 Am Ende dieses Kapitels 259

10 Sicherheitslücken aufspüren und ausnutzen 261

10.1 Fuzzing 261
10.1.1 Der einfachste Fuzzing-Test 262
10.1.2 Mutations-Fuzzer 262
10.1.3 Testdatensätze generieren 263
10.2 Sicherheitslücken untersuchen 264
10.2.1 Debugging von Anwendungen 264
10.2.2 Die Chancen erhöhen, um die Hauptursache für einen Absturz zu ermitteln 271
10.3 Gängige Sicherheitslücken ausnutzen 274
10.3.1 Exploit von Speicherlücken 275
10.3.2 Willkürliche Schreiboperationen 283
10.4 Shell-Code entwickeln 286
10.4.1 Erste Schritte 286
10.4.2 Einfache Debugging-Technik 289
10.4.3 Systemaufrufe ausführen 290
10.4.4 Andere Programme ausführen 295
10.4.5 Shell-Code mit Metasploit generieren 296
10.5 Maßnahmen gegen Speicherlücken 298
10.5.1 Data Execution Prevention 298
10.5.2 Return-Oriented Programming 300
10.5.3 Address Space Layout Randomization (ASLR) 302
10.5.4 Stacküberläufe durch Canaries erkennen 305
10.6 Am Ende dieses Kapitels 309

Anhang

A Toolkit für die Netzwerkprotokoll-Analyse 311

A.1 Tools zum passiven Capturing und zur Analyse von Netzwerkprotokollen 311

A.1.1 Microsoft Message Analyzer 312
A.1.2 TCPDump und LibPCAP 313
A.1.3 Wireshark 314

A.2 Aktives Netzwerk-Capturing und Analyse 315

A.2.1 Canape 315
A.2.2 Canape Core 316
A.2.3 Mallory 316

A.3 Netzwerkkonnektivität und Protokolltests 316

A.3.1 Hping 316
A.3.2 Netcat 317
A.3.3 Nmap 317

A.4 Webanwendungen testen 318

A.4.1 Burp Suite 318
A.4.2 Zed Attack Proxy (ZAP) 319
A.4.3 Mitmproxy 319

A.5 Frameworks zum Fuzzing, zur Paketgenerierung und zur Entwicklung von Exploits 320

A.5.1 American Fuzzy Lop (AFL) 320
A.5.2 Kali Linux 321
A.5.3 Metasploit-Framework 321
A.5.4 Scapy 321
A.5.5 Sulley 322

A.6 Netzwerk-Spoofing und -Umleitung 322

A.6.1 DNSMasq 322
A.6.2 Ettercap 322

A.7 Reverse Engineering von Executables 323

A.7.1 Java Decompiler (JD) 323
A.7.2 IDA Pro 324
A.7.3 Hopper 325
A.7.4 ILSpy 325
A.7.5 .NET Reflector 326

Index 327

1 Netzwerk-Grundlagen

Um Netzwerkprotokolle angreifen zu können, müssen Sie die Grundlagen der Vernetzung von Computern kennen. Je besser Sie verstehen, wie gängige Netzwerke aufgebaut sind und funktionieren, desto einfacher können Sie dieses Wissen nutzen, um neue Protokolle zu erfassen, zu analysieren und auszunutzen.

Im Verlauf dieses Kapitels werde ich grundlegende Konzepte vorstellen, die Ihnen bei der Analyse von Netzwerkprotokollen tagtäglich begegnen. Außerdem schaffe ich auch die Voraussetzung für eine bestimmte Art des Denkens über Netzwerkprotokolle, die es einfacher macht, bisher unbekannte Sicherheitslücken während der Analyse zu entdecken.

1.1 Netzwerkarchitekturen und -protokolle

Wir wollen zuerst einige grundlegende Netzwerkbegriffe besprechen und uns die fundamentale Frage stellen: Was ist ein Netzwerk? Ein *Netzwerk* ist eine Gruppe von zwei oder mehr Computern, die miteinander verbunden sind, um Informationen zu teilen. Jedes mit dem Netzwerk verbundene Gerät wird gewöhnlich als *Knoten* (engl. Node) bezeichnet, um die Beschreibung auf eine größere Palette von Geräten anwenden zu können. Abbildung 1–1 zeigt ein sehr einfaches Beispiel.

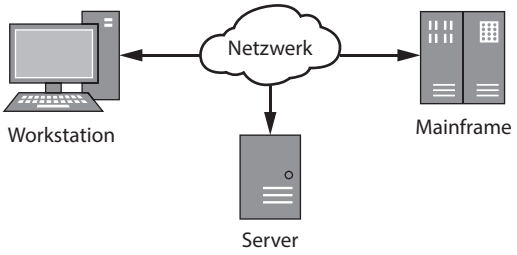


Abb. 1–1 *Einfaches Netzwerk mit drei Knoten*

Die Abbildung zeigt drei Knoten, die über ein gängiges Netzwerk miteinander verbunden sind. Jeder Knoten kann ein anderes Betriebssystem oder eine andere Hardware verwenden. Doch solange jeder Knoten einer Reihe von Regeln folgt, dem *Netzwerkprotokoll*, können sie mit jedem anderen Knoten des Netzwerks kommunizieren. Um sauber miteinander kommunizieren zu können, müssen alle Knoten im Netzwerk das gleiche Netzwerkprotokoll verstehen.

Ein Netzwerkprotokoll übernimmt viele Funktionen, dazu gehören eine oder mehrere der folgenden:

- **Verwaltung des Sessionzustands**
  Protokolle implementieren typischerweise Mechanismen, mit denen neue Verbindungen aufgebaut und vorhandene Verbindungen beendet werden können.
- **Identifizierung von Knoten durch Adressierung**
  Daten müssen im Netzwerk an den richtigen Knoten übertragen werden. Einige Protokolle implementieren einen Adressierungsmechanismus, um bestimmte Knoten oder Gruppen von Knoten zu identifizieren.
- **Flusssteuerung**
  Die Menge der über ein Netzwerk übertragenen Daten ist beschränkt. Protokolle können Wege zur Verwaltung des Datenflusses implementieren, um den Durchsatz zu erhöhen und die Latenz zu reduzieren.
- **Garantierte Reihenfolge der übertragenen Daten**
  Viele Netzwerke garantieren nicht, dass die Reihenfolge, in der die Daten gesendet werden, auch der Reihenfolge entspricht, in der sie eingehen. Ein Protokoll kann die Daten neu ordnen, um die Zustellung in der richtigen Reihenfolge sicherzustellen.
- **Erkennung und Korrektur von Fehlern**
  Viele Netzwerke sind nicht zu 100 Prozent zuverlässig, d.h., Daten können beschädigt werden. Es ist wichtig, Beschädigungen zu erkennen und (idealerweise) zu beheben.
- **Formatierung und Codierung von Daten**
  Daten liegen nicht immer in einem Format vor, das für die Übertragung in einem Netzwerk geeignet ist. Ein Protokoll kann Regeln zur Codierung von Daten festlegen, etwa die Codierung von Text in Binärdaten.

1.2 Die Internet-Protokoll-Suite

TCP/IP ist der von modernen Netzwerken verwendete De-facto-Protokollstandard. Obwohl Sie sich TCP/IP als ein Protokoll vorstellen können, ist es tatsächlich die Kombination von zwei Protokollen: dem *Transmission Control Protocol (TCP)* und dem *Internet Protocol (IP)*. Diese beiden Protokolle sind Teil der *Internet Protocol Suite (IPS)*, eines konzeptionellen Modells, das angibt, wie Netzwerkprotokolle Daten über das Internet senden. Es teilt die Netzwerkkomunikation, wie in Abbildung 1–2 zu sehen, in vier Schichten (Layer) auf.

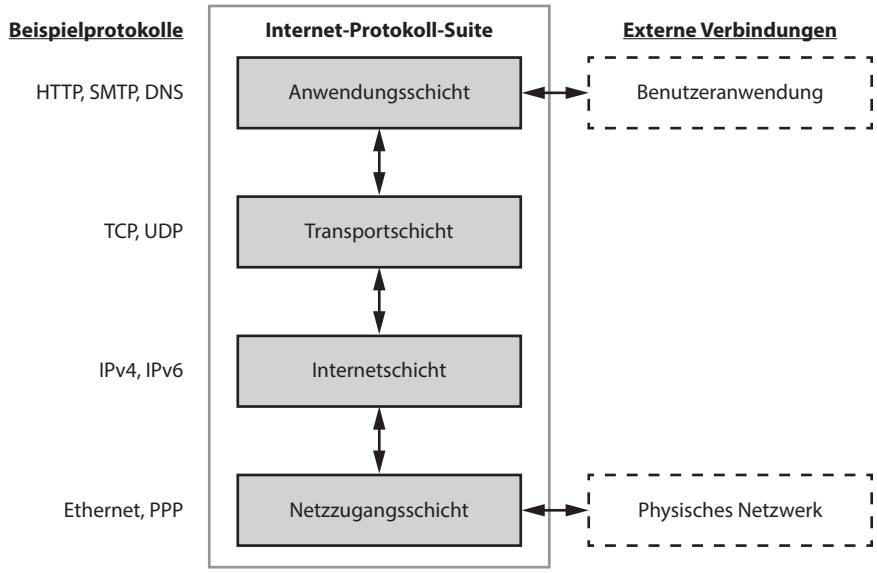


Abb. 1–2 *Schichten der Internet-Protokoll-Suite*

Diese vier Schichten bilden einen *Protokollstack*. Die folgende Liste erläutert jede Schicht der IPS:

- **Netzzugangsschicht (Layer 1)**
  Diese Schicht liegt auf der untersten Ebene und beschreibt die physikalischen Mechanismen, mit denen Informationen zwischen den Knoten eines lokalen Netzwerks übertragen werden. Bekannte Beispiele sind Ethernet (kabelgebunden und drahtlos) und das Point-to-Point-Protokoll (PPP).
- **Internetschicht (Layer 2)**
  Diese Schicht stellt die Mechanismen zur Adressiung der Netzwerkknoten bereit. Im Gegensatz zur Netzzugangsschicht müssen die Knoten nicht im gleichen Netzwerk liegen. Diese Schicht enthält das IP. In modernen Netzwerken kann das verwendete Protokoll die Version 4 (IPv4) oder die Version 6 (IPv6) sein.

- **Transportschicht (Layer 3)**
  Diese Schicht ist für die Verbindungen zwischen Clients und Servern verantwortlich. Manchmal stellt sie auch die korrekte Reihenfolge der Pakete sicher oder bietet das Multiplexing von Diensten an. Das Multiplexing von Diensten erlaubt es einem einzelnen Knoten, unterschiedliche Dienste zu unterstützen, indem jedem Service eine andere Nummer zugewiesen wird. Diese Nummer wird als *Port* bezeichnet. TCP und UDP (User Datagram Protocol) arbeiten auf dieser Schicht.
- **Anwendungsschicht (Layer 4)**
  Auf dieser Schicht sind Netzwerkprotokolle wie das *HyperText Transport Protocol (HTTP*, zur Übertragung von Webseiten), das *Simple Mail Transport Protocol (SMTP*, zur Übertragung von E-Mails) und das *Domain Name System (DNS,* zur Umwandlung von Namen in Adressen) angesiedelt. In diesem Buch konzentrieren wir uns hauptsächlich auf diese Schicht.

Jede Schicht interagiert nur mit der direkt über oder unter ihr liegenden Schicht, doch es muss auch externe Interaktionen mit dem Stack geben. Abbildung 1–2 zeigt zwei externe Verbindungen. Die Netzzugangsschicht interagiert mit einer physikalischen Netzwerkverbindung und überträgt Daten in ein physikalisches Medium wie Strom- oder Lichtimpulse. Die Anwendungsschicht interagiert mit der Anwendung: Eine *Anwendung* ist eine Sammlung zusammengehöriger Funktionalitäten, die dem Benutzer einen Dienst zur Verfügung stellen. Abbildung 1–3 zeigt beispielhaft eine Anwendung zur Verarbeitung von E-Mails. Der Dienst, der von der E-Mail-Anwendung angeboten wird, ist das Senden und Empfangen von Nachrichten über ein Netzwerk.

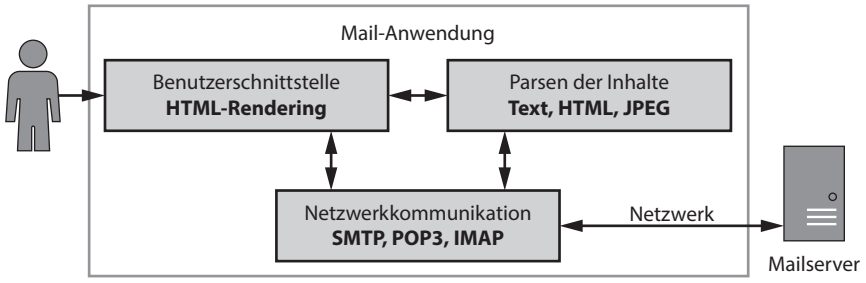


Abb. 1–3 *Beispielhafte E-Mail-Anwendung*

Typischerweise umfassen Anwendungen die folgenden Komponenten:

- **Netzwerkkommunikation**
  Diese Komponente kommuniziert über das Netzwerk und verarbeitet ein- und ausgehende Daten. Bei einer E-Mail-Anwendung läuft die Netzwerkkommunikation meist über ein Standardprotokoll wie SMTP oder POP3.
- **Parsen der Inhalte**
  Über ein Netzwerk transferierte Daten müssen üblicherweise extrahiert und verarbeitet werden (Parsen). Bei den Inhalten kann es sich um Textdaten (z.B. den Text der E-Mail), Bilder oder Videos handeln.
- **Benutzerschnittstelle**
  Die Benutzerschnittstelle (User Interface, kurz UI) erlaubt es dem Benutzer, empfangene E-Mails anzusehen und neue E-Mails zu verfassen bzw. zu senden. Bei einer E-Mail-Anwendung könnte die UI E-Mails mittels HTML in einem Webbrowser darstellen.

Beachten Sie, dass der Benutzer, der mit der UI interagiert, kein Mensch sein muss. Es kann sich auch um eine andere Anwendung handeln, die das Senden und Empfangen von E-Mails (z.B. über ein Kommandozeilen-Tool) automatisiert.

1.3 Datenkapselung

Jede Schicht der IPS baut auf der darunterliegenden Schicht auf und jede Schicht ist in der Lage, Daten der darüberliegenden Schicht zu kapseln, sodass sie zwischen den Schichten bewegt werden können. Die von jeder Schicht übertragenen Daten werden *Protocol Data Unit (PDU)* genannt.

1.3.1 Header, Footer und Adressen

Die PDU jeder Schicht enthält die zu übertragenden Nutzdaten. Üblicherweise stellt man den Nutzdaten einen *Header* voran, der für die Übertragung der Nutzdaten benötigte Informationen enthält, wie z.B. die *Adressen* der Quell- und Zielknoten. Manchmal besitzt eine PDU auch einen *Footer,* der an die Nutzdaten angehängt wird und Werte enthält, die eine korrekte Übertragung sicherstellen, etwa Prüfsummen. Abbildung 1–4 zeigt, wie die PDUs der IPS ausgelegt sind.

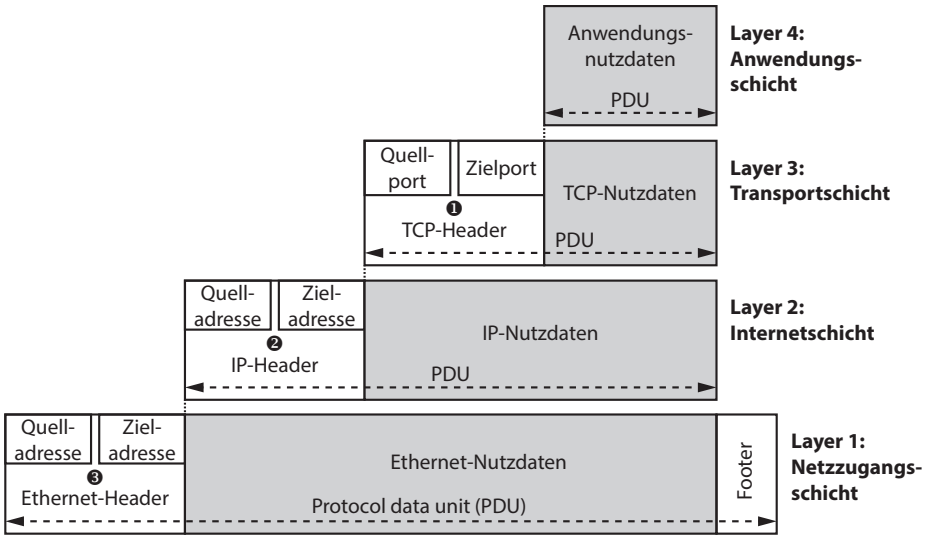


Abb. 1–4 *IPS-Datenkapselung*

Der TCP-Header enthält den Quell- und Zielport ❶. Diese Portnummern erlauben es einem einzelnen Knoten, mehrere Netzverbindungen aufzubauen. Die Portnummern für TCP (und UDP) reichen von 0 bis 65535. Die meisten Portnummern werden neuen Verbindungen nach Bedarf zugewiesen, doch einigen Nummern wurden bestimmte Dienste zugeordnet, wie etwa Port 80 für HTTP. (Eine aktuelle Liste der zugewiesenen Portnummern finden Sie bei den meisten unixoiden Betriebssystemen in der Datei */etc/services*.) Die TCP-Nutzdaten und den Header nennt man üblicherweise ein *Segment*, während UDP-Nutzdaten und -Header als *Datagramm* bezeichnet werden.

Das IP-Protokoll verwendet eine Quell- und eine Zieladresse ❷. Die *Zieladresse* erlaubt das Senden der Daten an einen bestimmten Knoten im Netzwerk. Über die *Quelladresse* weiß der Empfänger, welcher Knoten ihm Daten gesendet hat, was es ihm ermöglicht, dem Sender zu antworten.

IPv4 verwendet 32-Bit-Adressen, die üblicherweise als vier durch Punkte getrennte Zahlen wie z.B. 192.168.10.1 dargestellt werden. IPv6 nutzt 128-Bit-Adressen, weil 32-Bit-Adressen für die Anzahl der Knoten in modernen Netzwerken nicht mehr ausreichen. IPv6-Adressen werden üblicherweise als durch Doppelpunkte getrennte Hexadezimalzahlen wie z.B. fe80:0000:0000:0000:897b:581e:44b0:2057 geschrieben. Lange Folgen von 0000-Werten werden durch zwei Doppelpunkte ersetzt, d.h., die obige IPv6-Adresse kann auch als fe80::897b:581e:44b0:2057 geschrieben werden. IP-Nutzdaten und -Header werden üblicherweise als Paket (*packet)* bezeichnet.

Ethernet enthält ebenfalls Quell- und Zieladressen ❸. Ethernet verwendet einen als MAC-Adresse (*Media Access Control)* bezeichneten 64-Bit-Wert, der normalerweise bei der Produktion des Ethernet-Adapters festgelegt wird. MAC-

Adressen werden üblicherweise als Folge von durch Minuszeichen oder Doppelpunkte getrennten Hexadezimalzahlen wie 0A-00-27-00-00-0E dargestellt. Die Ethernet-Nutzdaten, zusammen mit dem Header und dem Footer, werden üblicherweise als *Frame* bezeichnet.

1.3.2 Datenübertragung

Sehen wir uns kurz an, wie beim IPS-Modell der Datenkapselung Daten von einem Knoten zum anderen übertragen werden. Abbildung 1–5 zeigt ein einfaches Ethernet-Netzwerk mit drei Knoten.

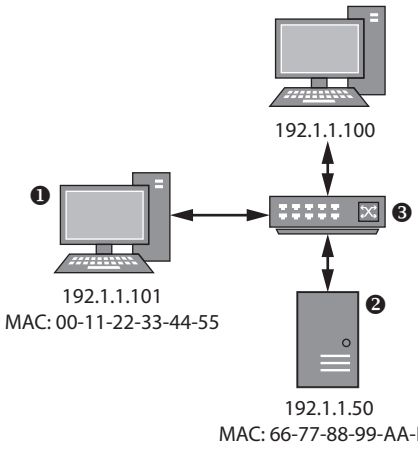


Abb. 1–5 *Einfaches Ethernet-Netzwerk*

In diesem Beispiel möchte der Knoten ❶ mit der IP-Adresse 192.1.1.101 Daten per IP-Protokoll an den Knoten ❷ mit der IP-Adresse 192.1.1.50 senden. (Der *Switch* ❸ leitet Ethernet-Frames zwischen allen Netzwerkknoten weiter. (Der Switch benötigt keine IP-Adresse, da er nur auf der Netzzugangsschicht arbeitet.) Wenn Daten zwischen den beiden Knoten gesendet werden sollen, passiert Folgendes:

1. Der Netzwerkstack des Betriebssystems ❶ kapselt die Daten der Anwendungs- und der Transportschicht und erzeugt ein IP-Paket mit der Quelladresse 192.1.1.101 und der Zieladresse 192.1.1.50.
2. An diesem Punkt kann das Betriebssystem die IP-Daten in einem Ethernet-Frame kapseln, kennt möglicherweise aber nicht die MAC-Adresse des Zielknotens. Es kann die MAC-Adresse für eine bestimmte IP-Adresse über das Address Resolution Protocol (ARP) anfordern, das einen Request an alle Knoten im Netzwerk sendet, um die MAC-Adresse für die IP-Adresse des Ziels zu ermitteln.

3. Sobald der Knoten an ❶ die ARP-Antwort erhält, kann er den Frame erzeugen, wobei die Quelladresse mit der lokalen MAC-Adresse 00-11-22-33-44-55 und die Zieladresse mit 66-77-88-99-AA-BB angegeben wird. Der neue Frame wird in das Netzwerk übertragen und vom Switch ❸ empfangen.
4. Der Switch leitet den Frame an den Zielknoten weiter, der das IP-Paket entpackt und prüft, ob die Ziel-IP-Adresse stimmt. Dann werden die IP-Nutzdaten entpackt und im Stack weitergeleitet, um von der wartenden Anwendung empfangen zu werden.

1.4 Netzwerk-Routing

Ethernet verlangt, dass alle Knoten direkt mit dem gleichen Netzwerk verbunden sind. Diese Anforderung ist für ein wirklich globales Netzwerk eine wesentliche Einschränkung, da es praktisch unmöglich ist, physikalisch jeden Knoten mit jedem anderen Knoten zu verbinden. Statt also alle Knoten direkt miteinander verbinden zu müssen, erlauben es die Quell- und Zieladressen, dass Daten über verschiedene Netzwerke *gerouted* (weitergeleitet) werden, bis sie den gewünschten Zielknoten erreichen (siehe Abb. 1–6).

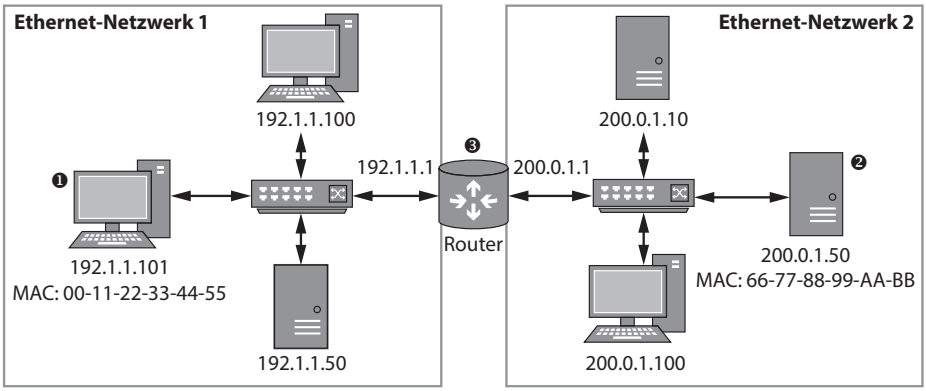


Abb. 1–6 *Routing zwischen zwei Ethernet-Netzwerken*

Abbildung 1–6 zeigt zwei Ethernet-Netzwerke mit eigenen IP-Adressbereichen. Die folgende Beschreibung erklärt, wie das IP-Protokoll dieses Modell nutzt, um Daten von dem Knoten bei ❶ in Netzwerk 1 an den Knoten bei ❷ in Netzwerk 2 zu senden.

1. Der Netzwerkstack des Betriebssystems auf dem Knoten bei ❶ kapselt die Daten der Anwendungs- und Transportschicht und baut ein IP-Paket mit der Quelladresse 192.1.1.101 und der Zieladresse 200.0.1.50 auf.
2. Der Netzwerkstack muss einen Ethernet-Frame senden, da aber die IP-Zieladresse in keinem Ethernet-Netzwerk existiert, mit dem der Knoten verbunden

ist, befragt der Netzwerkstack die *Routing-Tabelle* des Betriebssystems. In diesem Beispiel enthält die Routing-Tabelle einen Eintrag für die IP-Adresse 200.0.1.50. Dieser Eintrag zeigt an, dass der Router ❸ an IP-Adresse 192.1.1.1 weiß, wie man zu dieser Zieladresse gelangt.

3. Das Betriebssystem nutzt ARP, um die MAC-Adresse des Routers an 192.1.1.1 nachzuschlagen und das Original-IP-Paket wird im Ethernet-Frame mit dieser MAC-Adresse gekapselt.
4. Der Router empfängt den Ethernet-Frame und entpackt das IP-Paket. Wenn er die IP-Zieladresse prüft, erkennt er, dass dieses IP-Paket nicht für diesen Router, sondern für einen Knoten in einem anderen Netzwerk gedacht ist. Der Router schlägt die MAC-Adresse für 200.0.1.50 nach, kapselt das ursprüngliche IP-Paket in einem neuen Ethernet-Frame und sendet diesen an Netzwerk 2.
5. Der Zielknoten empfängt den Ethernet-Frame, entpackt das IP-Paket und verarbeitet dessen Inhalt.

Dieser Routing-Prozess kann sich mehrfach wiederholen. Ist der Router beispielsweise nicht direkt mit dem Netzwerk verbunden, das den Knoten 200.0.1.50 enthält, würde er seine eigene Routing-Tabelle konsultieren und den nächsten Router bestimmen, an den er das IP-Paket sendet.

Natürlich ist es praktisch nicht möglich, dass jeder Knoten im Netzwerk weiß, wie er zu jedem anderen Knoten im Internet gelangt. Wenn es für ein Ziel keinen expliziten Routing-Eintrag gibt, stellt die Routing-Tabelle einen Standardeintrag bereit, das sogenannte *Default Gateway*, der die IP-Adresse eines Routers enthält, der IP-Pakete an ihr Ziel weiterleiten kann.

1.5 Mein Modell für die Analyse von Netzwerkprotokollen

Die IPS beschreibt, wie die Netzwerkkommunikation funktioniert. Für Analysezwecke ist ein Großteil des IPS-Modells aber nicht relevant. Es ist einfacher, mein Modell zu nutzen, um das Verhalten eines Anwendungsprotokolls zu verstehen. Mein Modell besteht aus drei Schichten. Abbildung 1–7 zeigt diese Schichten und verdeutlicht, wie ich einen HTTP-Request analysieren würde.

Hier die drei Schichten meines Modells:

- **Inhaltsschicht (Content Layer)**
  Gibt den »Sinn« dessen wieder, was kommuniziert wird. In Abbildung 1–7 besteht der Sinn darin, mit einem HTTP-Request die Datei *image.jpg* abzurufen.
- **Codierungsschicht (Encoding Layer)**
  Legt die Regeln fest, nach denen der Inhalt repräsentiert werden soll. In diesem Beispiel wird die HTTP-Anfrage als HTTP-GET-Request codiert, der die abzurufende Datei festlegt.

- **Transportschicht (Transport Layer)**
  Legt die Regeln fest, nach denen die Daten zwischen den Knoten übertragen werden. In diesem Beispiel wird der HTTP-GET-Request über eine TCP/IP-Verbindung mit Port 80 des entfernten Knotens durchgeführt.

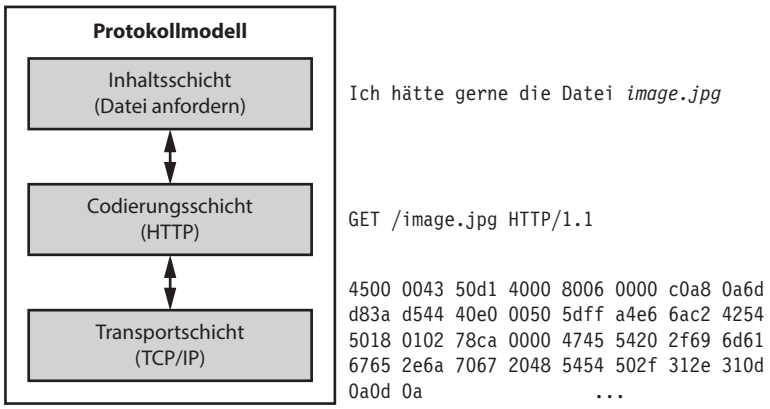


Abb. 1–7 *Mein Modellkonzept für Protokolle*

Diese Art der Aufteilung des Modells reduziert die Komplexität anwendungsspezifischer Protokolle, weil wir die Teile des Netzwerkprotokolls herausfiltern können, die für uns nicht relevant sind. Da es uns beispielsweise nicht interessiert, wie TCP/IP an den entfernten Knoten gesendet wird (wir gehen einfach davon aus, dass es irgendwie funktioniert), können wir TCP/IP-Daten als binären Transport betrachten, der schlicht funktioniert.

Um zu verstehen, warum dieses Protokollmodell nützlich ist, stellen Sie sich einfach vor, Sie müssen den Netzwerkverkehr irgendeiner Malware untersuchen. Sie finden heraus, dass die Malware HTTP nutzt, um Befehle vom Operator über einen Server zu empfangen. Der Operator könnte die Malware zum Beispiel anweisen, alle Dateien auf der Festplatte des infizierten Computers aufzulisten. Die Dateiliste kann an den Server zurückgeschickt werden und der Operator kann dann den Upload einer bestimmten Datei anfordern.

Wenn wir das Protokoll aus dem Blickwinkel betrachten, wie der Operator mit der Malware interagiert, indem er z.B. den Upload einer Datei veranlasst, können wir das neue Protokoll in die in Abbildung 1–8 aufgeführten Schichten aufteilen.

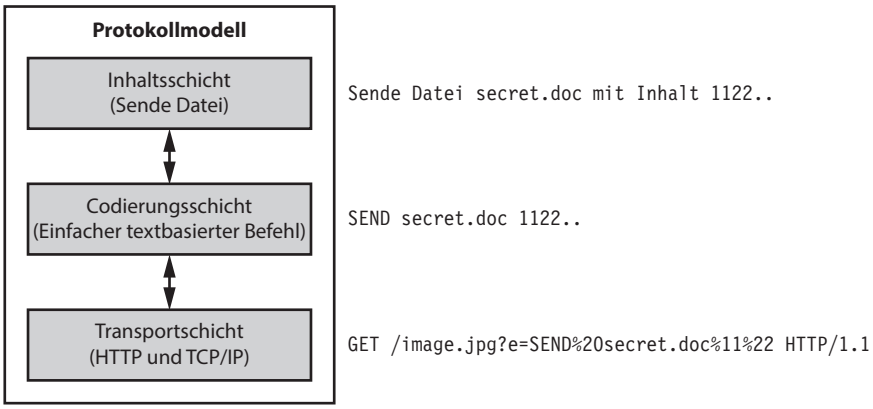


Abb. 1–8 *Modellkonzept für ein HTTP nutzendes Malware-Protokoll*

Die folgende Liste erläutert jede Schicht des neuen Protokollmodells:

- **Inhaltsschicht**
  Die bösartige Anwendung sendet eine gestohlene Datei namens *secret.doc* an den Server.
- **Codierungsschicht**
  Die Codierung des Befehls zum Senden der gestohlenen Datei besteht aus einem einfachen Textstring mit dem Befehl `SEND`, gefolgt vom Dateinamen und den Daten.
- **Transportschicht**
  Das Protokoll verwendet einen HTTP-Request-Parameter, um den Befehl zu übertragen. Es benutzt die übliche Prozentcodierung, um einen gültigen HTTP-Request zu erzeugen.

Beachten Sie, dass wir in diesem Beispiel nicht berücksichtigt haben, wie der HTTP-Request über TCP/IP gesendet wird. Wir haben die Codierungs- und Transportschicht aus Abbildung 1–7 in Abbildung 1–8 in der Transportschicht zusammengefasst. Zwar nutzt die Malware Low-Level-Protokolle wie TCP/IP, doch diese Protokolle sind nicht wichtig, wenn wir analysieren wollen, wie der Malware-Befehl eine Datei sendet. Das ist deshalb nicht wichtig, weil wir HTTP über TCP/IP als einzelne Transportschicht betrachten können, die einfach funktioniert, und uns lieber auf die Malware-Befehle konzentrieren wollen.

Indem wir unseren Blick auf die Schichten des Protokolls richten, die wir analysieren müssen, vermeiden wir viel Arbeit und können uns auf die wesentlichen Aspekte des Protokolls konzentrieren. Würden wir dieses Protokoll andererseits nach den Schichten aus Abbildung 1–7 analysieren, könnten wir annehmen, dass die Malware einfach die Datei *image.jpg* anfordert, weil es so aussieht, als wäre das alles, was der HTTP-Request macht.

1.6 Am Ende dieses Kapitels

Dieses Kapitel hat kurz in die Netzwerk-Grundlagen eingeführt. Ich habe die IPS vorgestellt sowie einige der Protokolle, denen Sie in echten Netzwerken begegnen werden. Außerdem habe ich gezeigt, wie Daten zwischen Knoten eines lokalen Netzwerks und über Router auch an entfernte Netzwerke übertragen werden. Darüber hinaus habe ich einen Weg beschrieben, Anwendungsprotokolle zu betrachten, der es Ihnen einfacher macht, sich auf die spezifischen Features des Protokolls zu konzentrieren und so die Analyse zu beschleunigen.

In Kapitel 2 werden wir diese Netzwerk-Grundlagen nutzen, um den Netzwerkverkehr für die Analyse zu erfassen, was man als Capturing bezeichnet. Das Ziel des Erfassens von Netzwerkverkehr besteht darin, auf die Daten zugreifen zu können, die Sie benötigen, um mit dem Analyseprozess zu beginnen, die verwendeten Protokolle zu identifizieren und letztlich die Sicherheitslücken aufzuspüren, die Sie ausnutzen können, um Anwendungen zu kompromittieren, die dieses Protokoll verwenden.

2
Capturing von Anwendungsverkehr

Überraschenderweise kann das Capturing, also das Erfassen nützlichen Verkehrs bei der Protokollanalyse, eine Herausforderung darstellen. Dieses Kapitel beschreibt zwei Aufzeichnungstechniken: *passives* und *aktives* Capturing. Passives Capturing interagiert nicht direkt mit dem Netzwerkverkehr. Stattdessen extrahiert es die Daten, während sie *über die Leitung laufen*, was Ihnen aus Tools wie Wireshark vertraut sein dürfte.

Sie werden sehen, dass unterschiedliche Anwendungen unterschiedliche Mechanismen (mit ihren jeweiligen Vor- und Nachteilen) verwenden, um Verkehr umzuleiten. Aktives Capturing greift in den Verkehr zwischen einer Clientanwendung und dem Server ein, was zwar sehr leistungsfähig ist, aber auch zu Komplikationen führen kann. Sie können sich aktives Capturing als eine Art Proxy, oder auch als Man-in-the-Middle-Angriff vorstellen. Sehen wir uns diese aktiven und passiven Techniken etwas genauer an.

2.1 Passives Capturing von Netzwerkverkehr

Passives Capturing ist eine relativ einfache Technik: Sie verlangt üblicherweise keine spezielle Hardware und Sie müssen auch keinen eigenen Code entwickeln. Abbildung 2–1 zeigt ein gängiges Szenario: Ein Client und ein Server kommunizieren per Ethernet über ein Netzwerk.