HANSER

Stille im Netz

Michael Zalewski

Ein Praxishandbuch zu passiver Reconnaissance und indirekten Angriffen

ISBN 3-446-40800-2

Leseprobe

Weitere Informationen oder Bestellungen unter http://www.hanser.de/3-446-40800-2 sowie im Buchhandel

6 Echos der Vergangenheit

Sechstes Kapitel. In welchem wir anhand eines skurrilen Fehlers in Ethernet lernen, wie wichtig es ist, deutlich zu sprechen

Im vorherigen Kapitel haben wir die Grundlagen der Ethernetkommunikation angerissen. Diese scheinbar narrensichere und dabei überraschend simple Methode ist offensichtlich nicht in der Lage, schwerwiegende Sicherheitsprobleme zu verursachen – vielleicht mit Ausnahme eines potenziellen Missbrauchs der Vertrauensbeziehung, der durch das reguläre Versenden der Daten an *alle* Punkte im Netzwerk (Broadcasting) möglich wird. Dies ist eine wohlbekannte und wohlverstandene Eigenschaft von Ethernetnetzwerken; entsprechende Abhilfemaßnahmen sind unter anderem der Einsatz von Switches und Bridges und die Netzwerksegmentierung.

Nichtsdestoweniger äußert sich dieser Aspekt auf völlig unvorhergesehene Art und Weise. Das liegt in erster Linie an der unglücklichen Wahl der Wörter (bzw. an ihrem Fehlen) in den offiziellen Implementierungsanforderungen für Ethernettreiber. Folge ist ein weitverbreitetes Implementierungsproblem, das inzwischen ein solches Ausmaß erreicht hat, dass es ein eigenes Kapitel in diesem Buch verdient hat. In diesem Kapitel erläutern wir eine interessante Fallstudie dieser Klasse von Problemen, an denen niemand schuld sein will.

6.1 Der Turmbau zu Babel

Das Ethernetprotokoll stellt alle erforderlichen Mittel zur Verfügung, um Bytes über eine Leitung zu verbreiten, nämlich ein maschinennahes Datenkodiersystem und ein Datenformat, das einen Teil der Daten aufnehmen kann. Der Ethernet-Frame enthält Versandanweisungen (d. h. wer die Daten sendet und wer als Empfänger vorgesehen ist) sowie eine kurze Beschreibung des gekapselten Datentyps. Weitere Methoden zur Fehlererkennung sind ebenfalls vorhanden. Der gesamte Frame wird dann an alle Systeme (einschließlich des

potenziellen Empfängers) geschickt. Funktionalitätsseitig ähnelt Ethernet den Datenkapselungssystemen über verschiedene Medien oder in verschiedenen Anwendungen wie etwa Frame Relay, ATM (Asynchronous Transfer Mode), PPP (Point-to-Point Protocol) usw.

Die Frage lautet: Welche Daten sollten durch einen solchen Ethernet-Frame übertragen werden? Computer verwenden Hunderte von Formaten und Anwendungsprotokollen und können unterschiedlichste Anwendungen von Netzwerkspielen und Chat-Clients bis hin zu wissenschaftlichen Simulationen ausführen. Aufgrund dessen ist es zwar möglich, aber nicht sinnvoll, die Daten für einen entfernten Empfänger einfach in einem Ethernet-Frame zu kapseln, denn der Empfänger würde gar nicht wissen, was er mit ihnen anfangen sollte: Handelt es sich um eine eingehende E-Mail? Ein Bild aus dem Internet? Oder vielleicht doch Konfigurationsdaten? Man weiß es nicht so genau. Die Unterscheidung wird zudem durch die Tatsache erschwert, dass ein normaler Computer eine Vielzahl von Programmen praktisch zeitgleich ablaufen lässt.

Ethernet wirft aber noch ein anderes Problem größeren Ausmaßes auf: Wie erreicht man den Gegenüber? Das Broadcasting von Daten an alle Punkte im lokalen Netzwerk ist einfach; was aber, wenn das andere System – der Punkt, den einer der lokalen Benutzer zu erreichen hofft – gar nicht lokal ist? Was, wenn er über ein WAN (Wide-Area Network) erreicht werden muss und ein ganz anderes Sicherungsschichtprotokoll verwendet? Sogar dann, wenn wir einen Weg finden, die Daten jenem entfernten Empfänger zukommen zu lassen, bleibt doch eine grundlegende Frage: Wie muss das Paket adressiert werden?

Ethernet verwendet ein eigenes, spezielles Adressierungsschema. Hosts werden anhand ihrer theoretisch eindeutigen Hardwareidentifikationsnummer – der MAC-Adresse (Media Access Control, Medienzugriffssteuerung) – kontaktiert, die vom Hersteller in jeden Ethernetadapter fest eingebrannt wird. Diese Adressen sind nur für Ethernet von Bedeutung; für andere Netzwerktypen spielen sie keine Rolle und sind zudem weitgehend ungeeignet, eine Hardwarekomponente zu orten, sofern Sie nicht gerade Zugriff auf die lokale Konfiguration haben. Hierdurch entsteht ein Problem mit der Vertrauenswürdigkeit: Wer beispielsweise hat eine Karte mit der MAC-Adresse 00:0D:56:E3:FB:E4 gekauft, und wo befinden sich Karte und Käufer jetzt? Können Sie wirklich darauf bauen, dass Sie es mit dem ursprünglichen Käufer und nicht mit einem Betrüger zu tun haben?

Maschinennahe Adressierungsschemata wie dieses sind für die Weiterleitung von Daten zum Empfänger normalerweise unbrauchbar, sofern die Hardware mit der jeweiligen MAC-Adresse nicht direkt an das physische Netzwerk des Absenders angeschlossen ist. Es gibt keine Möglichkeit, eine physische Gerätekennung einem Punkt auf diesem Globus direkt zuzuordnen und daraus einen Pfad zu erschließen, der für den Versand der Daten verwendet werden sollte.

6.1.1 Das OSI-Modell

Die Protokolle der Sicherungsschicht wurden entwickelt, um die Kommunikation zwischen lokalen Knoten bzw. – in Extremfällen – zwischen zwei festen Endpunkten einer gemeinsamen Verbindung zu ermöglichen. Um Netzwerkverbunde zu ermöglichen und

einige praktischere Anwendungen von Netzwerken praktikabel zu machen, wurde eine hierarchische Struktur der Netzwerkprotokolle entwickelt: das OSI-Modell.

Das in Abbildung 6.1 gezeigte *OSI-Modell (Open Systems Interconnection)* definiert die physische Verbindungsebene (oder *Bitübertragungsschicht*) als unterste Schicht und setzt Funktionen höherer Ebenen darauf auf. Protokolle auf Leitungsebene bilden die zweite Schicht (*Sicherungsschicht*) und sind erwartungsgemäß als Möglichkeit definiert, mit anderen lokalen Knoten zu kommunizieren, die dieselbe physische Verbindung nutzen. Diese Protokolle übertragen übergeordnete, nicht leitungsspezifische Protokolldaten, die der dritten Schicht des Modells – der *Vermittlungsschicht* – entstammen. Das IP-Protokoll (Internet Protocol) ist das wohl bekannteste Beispiel für ein solches Protokoll.

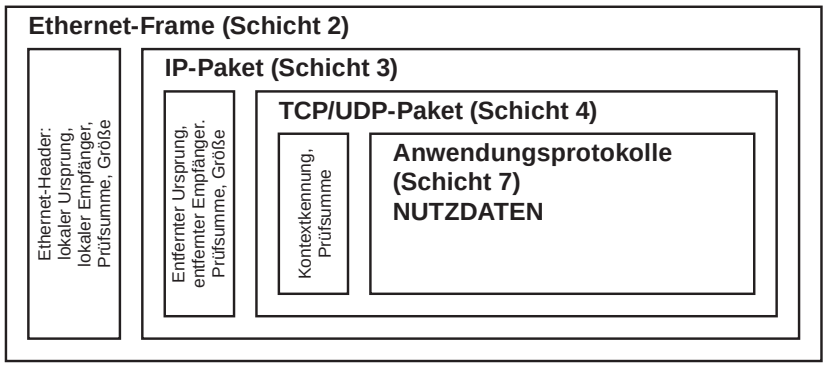

Abbildung 6.1 Physisches Datenlayout im OSI-Modell (Beispiel)

Die dritte Schicht vermittelt Informationen über das allgemeine Wesen des Datenverkehrs sowie universelle Kennungen von Ursprung und Endziel der Daten. Hier kommt eine netzwerkspezifische Adressierung zum Einsatz, die das Routing des Pakets erleichtert. Anders als bei Protokollen der zweiten Schicht werden die Daten der dritten Schicht unterwegs nicht gelöscht oder modifiziert und enthalten auch keine leitungsspezifischen Merkmale wie MAC-Adressen, CSMA/CD-Zusatzdaten usw.

Die vierte Schicht stellt ein Vehikel zur Herstellung bestimmter Kommunikationskanäle zwischen Endpunkten bereit, beginnend und endend auf einem gegebenen System. Auf diese Weise wird eine Möglichkeit zur gleichzeitigen Kommunikation mehrerer Typen und Kanäle vermittelt. Keines der Protokolle der vierten Schicht muss von Zwischensystemen verstanden werden, damit die Daten ordnungsgemäß beim Empfänger ausgeliefert werden können. Die Pakete werden nur vom endgültigen Empfänger interpretiert, der anhand ihrer bestimmt, welche Anwendung die Daten erhalten soll und wie sich das Datensegment zu den benachbarten Paketen verhält.

Die nächsten Schichten des OSI-Modells sind wahrscheinlich weniger interessant und verschmelzen tendenziell miteinander. Die fünfte Schicht soll Verlässlichkeitsfunktionen bereitstellen, die häufig entweder in den Protokollen der Schicht 4 wie etwa TCP/IP (Transmission Control Protocol/Internet Protocol) oder auf Anwendungsebene vorhanden sind. In

manchen Fällen sind sie noch nicht einmal implementiert (nämlich dann, wenn eine zuverlässige Kommunikation gar nicht erforderlich ist). Die sechste Schicht bietet „Bibliotheksfunktionen“ wie etwa Entkomprimierung und Dekodierung der Daten und wird wie auch Schicht 5 vorzugsweise über Funktionen auf der Anwendungsebene realisiert. Die siebte Schicht schließlich ist die *Anwendungsschicht*. Hier werden die Daten in ein spezifisches Format übertragen.

Beachten Sie, dass die oberen Schichten des OSI-Modells unabhängig von den unteren Schichten sind, denn sie beziehen sich auf die übertragenen Daten. Zum betreffenden Zeitpunkt lassen sich die unteren Schichten nach und nach entfernen, ohne dass die Daten oder die Fähigkeit, sie weiterzuverarbeiten, verloren gehen. Die zweite Schicht wird bei jedem Zwischensystem verworfen und neu erstellt, die dritte lässt sich entfernen, sobald die Daten beim Zielsystem angekommen sind. Schicht 4 wird vor dem Übergeben der Daten an die Clientanwendung gelöscht.

Die Vermittlungsschicht (Schicht 3) bleibt normalerweise vollkommen unabhängig vom darunter vorhandenen Sicherungsschichtprotokoll; sie enthält vollständige Angaben zu Absender und Empfänger, einen Mechanismus für den Integritätsschutz (Prüfsummenbildung) und Informationen zur Größe der übertragenen Nutzdaten. Dies ist genau das, was IP tut.

Eine wichtige Auswirkung dieser Konstruktion besteht darin, dass alle überflüssigen Daten, die während des Transports in Schicht 2 an das Paket angehängt werden, die Art und Weise, wie der Adressat die IP-Angaben interpretiert, nicht beeinträchtigen.

6.2 Der fehlende Satz

Im vorigen Kapitel erwähnte ich bei der Beschreibung von Ethernet eine interessante Anforderung, die sich aus der Notwendigkeit ergibt, den Jam-Code im Falle einer Kollision zuverlässig im Netzwerk verschicken zu können: die *Mindestlänge* eines Ethernet-Frames.

Diese Anforderung wurde auch Bestandteil der offiziellen Kapselungsspezifikationen für IP über Ethernet (z. B. RFC1042, „A Standard for the Transit of Internet Protocol Datagrams Over IEEE 802 Networks“[1]), die vorsehen, dass Frames, die die erforderliche Mindestlänge unterschreiten, aufgefüllt werden. Dieses Auffüllen (auch *Padding* genannt) kann nach Bedarf ausgeführt werden und wirkt sich nicht auf die in der IP-Schicht übertragenen Daten aus, da sich die in den IP-Headern angegebene Paketlänge nicht ändert. Insofern werden die Fülldaten vom Empfänger nicht als Teil der Daten übergeordneter OSI-Schichten interpretiert.

Es gibt allerdings ein kleines Problem: Zwar erfordert der RFC das Auffüllen der Daten mit Nullen, er gibt aber nicht an, wer die Fülldaten richten oder einfügen und auf welcher Softwareebene das Auffüllen erfolgen soll. Die Notwendigkeit, dass die Fülldaten einen

[1] Post88

bestimmten Wert haben müssen, ist auch eine Anforderung, die ihrem Wesen nach ziemlich willkürlich ist. Insofern wird ihr keine Aufmerksamkeit zuteil: Hätten sie andere Werte, dann würde sich das nicht auf die Funktionsweise des Protokolls auswirken, weil die überzähligen Daten beim Empfang einfach verworfen werden.

Um die Verwirrung noch zu steigern, bieten viele Netzwerkkarten eine so genannte *Autopadding-Funktion*, die vom Betriebssystem an die Hardware gesendete Pakete ergänzt, wenn diese zu kurz geraten sind; die Funktion prüft aber nicht den eigentlichen Inhalt der Fülldaten, wenn das erforderliche Auffüllen bereits softwareseitig erfolgt ist. Dies hat aufseiten einiger Entwickler zu viel Konfusion geführt, denn diese entschieden, dass die Größenanforderung zu beachten sei, und änderten die Größe eines Pakets in der Software, indem sie einfach die deklarierte Länge erweiterten. Häufig bemerkten sie nicht, dass die Daten zwischen dem Ende des IP-Pakets und dem des aufgefüllten Frames weder vom Treiber noch vom Betriebssystem oder der Hardware initialisiert (d. h. auf 0 gesetzt) wurden.

Dieser Umstand blieb jahrelang weitgehend unbemerkt, obwohl er eine knifflige Frage aufwarf, die Netzwerkhacker regelmäßig in den Wahnsinn trieb: Die Pakete, die sie aus lokalen Systemen abriefen, enthielten an ihrem Ende häufig eine Menge irrelevanten Datenmülls wie etwa Fragmente von Webseiteninhalten oder sogar Chatprotokollen. Die Hacker machten die Empfängerseite dafür verantwortlich – Hardware, Anwendungen zur Datenverkehrsanalyse oder Bibliotheken seien fehlerhaft –, aber schließlich gaben sie es auf, nach der Ursache zu suchen, denn die Relevanz dieses Sachverhaltes war vernachlässigbar. Und so erhielt das Problem niemals die Aufmerksamkeit, die es eigentlich verdient hätte.

Zumindest nicht, bis Ofir Arkin und Josh Anderson von @Stake sich die Sache genauer ansahen. Ihr im Jahr 2003 erschienener Beitrag „EtherLeak: Ethernet Frame Padding Information Leaks“[2] untersucht diesen Aspekt im Detail. Die Autoren stellten fest, dass bei einer großen Zahl marktführender Systeme – z. B. Linux, NetBSD, Microsoft Windows und anderen Plattformen – am Ende eines neu erstellten Ethernet-Frames keine Initialisierung des Speichers durchgeführt wird, nachdem seine Länge geändert wurde. Einigen Implementierungen gelingt es noch nicht einmal, die Größe des Frames korrekt abzuändern oder eine passende Anzahl von Bytes an die Hardwareschicht zu senden.

Aufgrund dessen wird das IP-Paket mit den Daten aufgefüllt, die gerade im betreffenden Bereich des Systemspeichers vorhanden sind und zuvor eigentlich für andere Zwecke dort abgelegt wurden. Der Speicher kann also je nach Treiber oder Betriebssystem entweder einen Teil eines zuvor gesendeten Pakets oder ein anderes Fragment des Kernel-Speichers enthalten. Und an dieser Stelle entsteht ein faszinierendes Datenenthüllungsszenario: Ein Angreifer sendet unscheinbare und zulässige Daten an das Opfer und erhält mit etwas Glück potenziell sensible Daten zurück. Der Umfang der preisgegebenen Daten ist in der Regel ausreichend, um Besorgnis auszulösen.

Die Schwachstelle beschränkt sich auf ein einzelnes Ethernetnetzwerk und ist insofern örtlich relativ beschränkt und in einer typischen LAN-Umgebung unkritisch. Es bleibt aber

[2] Arki03

ein Problem von einiger Bedeutung, und auch wenn jedes lokale Netzwerk teilweise anfällig für Schnüffeleien ist, so gestattet dieses Problem doch einige Schlussfolgerungen, die über den Rahmen des Offensichtlichen hinausgehen:

- Bei Systemen, die für Ethernet-Frames dynamische Puffer verwenden (z. B. Linux), können in den Fülldaten nicht nur Teile des vorherigen Frames auftauchen, sondern auch Fragmente anderer Speicherinhalte, z. B. angezeigte oder bearbeitete Dokumente, URLs, Passwörter oder andere sensible Daten. In diesem Fall kann ein sorgfältiger Beobachter unter Umständen auf Informationen zugreifen, die sich auf andere Weise nicht im Netzwerk erschleichen lassen.
- Bei Systemen, die statische Puffer nur zur Vorbereitung der Ethernet-Frames verwenden, lässt sich das Problem nutzen, um in Systeme einzudringen, die gegen Sniffer geschützt sind (z. B. Switches). Auf diese Weise kann der Angreifer Daten einer anderen Verbindung abfangen.
- Bei bestimmten statischen Pufferkonstruktionen werden unter Umständen Daten eines anderen Segments auf einem Multihoming-System offengelegt, dessen eine Schnittstelle an ein allgemeines LAN angeschlossen ist, während die andere mit einem eingeschränkten Netzwerk verbunden ist. Auf diese Weise gelangen Teile möglicherweise geheimer Daten in die öffentliche Infrastruktur.

Die Autoren des Dokuments haben verschiedene Open-Source-Implementierungen umfassend überprüft und festgestellt, dass eine Vielzahl von Ansätzen und Pufferentwürfen eingesetzt wird und dass es kein vorherrschendes Reservierungs- und Verwendungskonzept für Puffer gibt. Und was schließen sie daraus? Dass eine typische Netzwerkumgebung früher oder später von allen drei Problemtypen betroffen sein wird.

6.3 Denkanstöße

Das hier beschriebene Problem ist für Ethernet und für das Netzdesign nicht ungewöhnlich: Solche Fälle treten meistens auf, wenn eine ansonsten sehr detaillierte Implementierungsdokumentation einen erforderlichen Schritt auslässt oder nur ungenau beschreibt. Viele Entwickler übersehen das Problem bei der Implementierung des Standards schlichtweg. Wären mehr derartig schwammige Anweisungen vorhanden, dann wäre ein Entwickler wahrscheinlich gezwungen, über das Problem nachzudenken; stattdessen implementiert er einfach Schrittanleitungen, weswegen er weitaus anfälliger dafür ist, einen Fehler zu begehen. „Narrensichere“ Anweisungen, die beschreiben, wie bestimmte Aufgaben durchzuführen sind – und nicht, welche Ziele erreicht werden sollen –, gehen oft nach hinten los. Wir werden – wenn auch in einem etwas anderem Kontext – in Teil III dieses Buches noch einmal auf das Problem protokollspezifischer Datenlecks zurückkommen.