WIRESHARK FOR SECURITY PROFESSIONALS
Using Wireshark and the Metasploit Framework

Jessey Bullock
with Jeff T. Parker

WILEY
To my loving wife Heidi, my family, friends, and all those I have had the opportunity to learn from. —Jessey

To Mom. Thank you. —Jeff
Credits

Project Editor
John Sleeva

Technical Editor
Rob Shimonski

Production Editor
Athiyappan Lalith Kumar

Copy Editor
Kim Heusel

Production Manager
Katie Wisor

Manager of Content Development and Assembly
Mary Beth Wakefield

Marketing Manager
Carrie Sherrill

Professional Technology and Strategy Director
Barry Pruett

Business Manager
Amy Knies

Executive Editor
Jim Minatel

Project Coordinator, Cover
Brent Savage

Proofreader
Nancy Bell

Indexer
Nancy Guenther

Cover Designer
Wiley

Cover Image
© Jonathan Haste/iStockPhoto
Jessey Bullock is a security engineer with a diverse background, having worked both as a security consultant and as an internal security team member. Jessey started out supporting network administration while trying to break into the security industry, and Wireshark has always been an integral part of his tool set. His varied skill set was honed across numerous industries, such as energy and finance, even having worked for a gaming company.

Jessey’s experience includes a deep understanding of offensive and application security. As a consultant, Jessey performed engagements involving everything from incident response to embedded device testing. Jessey currently focuses on application security and has a keen interest in scaling security testing while providing day to day security support for developers and performing assessments of internally developed products.

In his free time, Jessey enjoys gaming with his son, writing the occasional Python code, and playing grumpy sysadmin for his wife’s restaurant business.

Jeff T. Parker is a seasoned security professional and technical writer. His 20 years of experience began with Digital Equipment Corporation, then on to Compaq and Hewlett Packard, where Jeff primarily consulted on complex enterprise environments. During the HP years, Jeff shifted his focus from systems to security. Only IT security has matched an insatiable appetite for learning and sharing.

Having done the “get as many certifications as you can” phase, Jeff is most proud of his service to clients, including UN agencies, government services, and enterprise corporations.

Jeff holds degrees in subjects far from IT, yet he only makes time to hack away at his home lab. He and his family enjoy life in Halifax, Nova Scotia, Canada.

Most excitedly, Jeff timed this project’s end with a much-anticipated new project: house training a new puppy.
About the Technical Editor

Rob Shimonski (www.shimonski.com) is a best-selling author and editor with more than 20 years of experience developing, producing, and distributing print media in the form of books, magazines, and periodicals, and more than 25 years working in the Information Technology field. To date, Rob has successfully helped create, as both an author and an editor, more than 100 books that are currently in circulation. Rob has an extremely diverse background in the print media industry, filling roles such as author, co-author, technical editor, copy editor, and developmental editor. Rob has worked for countless companies, including CompTIA, Cisco, Microsoft, Wiley, McGraw Hill Education, Pearson, the National Security Agency, and the US military.

As a Wireshark guru, Rob’s experience goes back to the beginning of the application’s existence. Having worked with Ethereal and various other packet capturing tools, Rob has been at the forefront of watching Wireshark evolve into the outstanding tool it is today. Rob has also captured this evolution in various written works, including Sniffer Pro: Network Optimization and Troubleshooting Handbook (Syngress, 2002) and The Wireshark Field Guide: Analyzing and Troubleshooting Network Traffic (Syngress, 2013). Rob has also worked with INE.com to create a practitioner and advanced practitioner video series detailing the usage and how to work with Wireshark in 2015. In 2016, Rob focused his energies on helping other authors develop their works to ensure technical accuracy in advanced topics within the Wireshark toolset. Rob is also certified as both a Wireshark Certified Network Analyst (WCNA) and a Sniffer Pro SCP.
This book owes a big thank you to the awesome developers of the Wireshark suite, as well as the developers of Metasploit, Lua, Docker, Python, and all the other open-source developers who make amazing technology accessible. Thanks also to the people at Wiley for putting up with me, especially John Sleeva and Jim Minatel, and to Rob Shimonski, the fantastic technical editor who helped keep the book correct and useful. Special thanks go to my co-author Jeff Parker for taking on the challenge of writing this book. He was a blast to work with and is owed immense credit for helping make this book possible.

I would also like to thank Jan Kadijk, John Heasman, Jeremy Powell, Tony Cargile, Adam Matthews, Shaun Jones, and Connor Kennedy for contributing ideas and support.

—Jessey

Kudos to the Wiley team, including Jim Minatel, John Sleeva, and Kim Heusel, for their dedication to carry this book to the finish line. Big thanks to Rob Shimonski, the technical editor, who performed with great patience to ensure we left no gaps or confusion.

To Jessey, the book’s visionary and the W4SP Lab guru, I thank you for being ever gracious and collaborative. All your effort concludes with a book and online resources that we can both be proud of.
To Carole Jelen, my literary agent in sunny southern California, all opportunities start with you. You are an endless provider of growth and have my deep gratitude. Thanks, Carole!

The biggest thanks go to my wife and my best friend. I’m grateful for her patience and support. To our two kids, Dad is back and ready to play (and research for the next book—wink, wink).

—Jeff
## Contents

**Introduction**

<table>
<thead>
<tr>
<th>Chapter 1</th>
<th><strong>Introducing Wireshark</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>What Is Wireshark? 2</td>
</tr>
<tr>
<td></td>
<td>A Best Time to Use Wireshark? 2</td>
</tr>
<tr>
<td></td>
<td>Avoiding Being Overwhelmed 3</td>
</tr>
<tr>
<td></td>
<td>The Wireshark User Interface 3</td>
</tr>
<tr>
<td></td>
<td>Packet List Pane 5</td>
</tr>
<tr>
<td></td>
<td>Packet Details Pane 6</td>
</tr>
<tr>
<td></td>
<td>Packet Bytes Pane 8</td>
</tr>
<tr>
<td></td>
<td>Filters 9</td>
</tr>
<tr>
<td></td>
<td>Capture Filters 9</td>
</tr>
<tr>
<td></td>
<td>Display Filters 13</td>
</tr>
<tr>
<td></td>
<td>Summary 17</td>
</tr>
<tr>
<td></td>
<td>Exercises 18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 2</th>
<th><strong>Setting Up the Lab</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Kali Linux 20</td>
</tr>
<tr>
<td></td>
<td>Virtualization 22</td>
</tr>
<tr>
<td></td>
<td>Basic Terminology and Concepts 23</td>
</tr>
<tr>
<td></td>
<td>Benefits of Virtualization 23</td>
</tr>
<tr>
<td></td>
<td>VirtualBox 24</td>
</tr>
<tr>
<td></td>
<td>Installing VirtualBox 24</td>
</tr>
<tr>
<td></td>
<td>Installing the VirtualBox Extension Pack 31</td>
</tr>
<tr>
<td></td>
<td>Creating a Kali Linux Virtual Machine 33</td>
</tr>
<tr>
<td></td>
<td>Installing Kali Linux 40</td>
</tr>
<tr>
<td></td>
<td>The W4SP Lab 46</td>
</tr>
<tr>
<td></td>
<td>Requirements 46</td>
</tr>
<tr>
<td></td>
<td>A Few Words about Docker 47</td>
</tr>
<tr>
<td></td>
<td>What Is GitHub? 48</td>
</tr>
</tbody>
</table>
Chapter 5  Diagnosing Attacks  129
   Attack Type: Man-in-the-Middle  130
      Why MitM Attacks Are Effective  130
      How MitM Attacks Get Done: ARP  131
      W4SP Lab: Performing an ARP MitM Attack  133
      W4SP Lab: Performing a DNS MitM Attack  141
      How to Prevent MitM Attacks  147
   Attack Type: Denial of Service  148
      Why DoS Attacks Are Effective  149
      How DoS Attacks Get Done  150
      How to Prevent DoS Attacks  155
   Attack Type: Advanced Persistent Threat  156
      Why APT Attacks Are Effective  156
      How APT Attacks Get Done  157
      Example APT Traffic in Wireshark  157
      How to Prevent APT Attacks  161
   Summary  162
   Exercises  162

Chapter 6  Offensive Wireshark  163
   Attack Methodology  163
   Reconnaissance Using Wireshark  165
   Evading IPS/IDS  168
      Session Splicing and Fragmentation  168
      Playing to the Host, Not the IDS  169
      Covering Tracks and Placing Backdoors  169
   Exploitation  170
      Setting Up the W4SP Lab with Metasploitable  171
      Launching Metasploit Console  171
      VSFTP Exploit  172
      Debugging with Wireshark  173
      Shell in Wireshark  175
      TCP Stream Showing a Bind Shell  176
      TCP Stream Showing a Reverse Shell  183
      Starting ELK  188
   Remote Capture over SSH  190
   Summary  191
   Exercises  192

Chapter 7  Decrypting TLS, Capturing USB, Keyloggers,
           and Network Graphing  193
   Decrypting SSL/TLS  193
      Decrypting SSL/TLS Using Private Keys  195
Chapter 8  Scripting with Lua 221

Why Lua? 222
Scripting Basics 223
  Variables 225
  Functions and Blocks 226
  Loops 228
  Conditionals 230
Setup 230
  Checking for Lua Support 231
  Lua Initialization 232
  Windows Setup 233
  Linux Setup 233
Tools 234
  Hello World with TShark 236
  Counting Packets Script 237
  ARP Cache Script 241
Creating Dissectors for Wireshark 244
  Dissector Types 245
  Why a Dissector Is Needed 245
  Experiment 253
Extending Wireshark 255
  Packet Direction Script 255
  Marking Suspicious Script 257
  Snooping SMB File Transfers 260
Summary 262

Index 265
Welcome to *Wireshark for Security Professionals*. This was an exciting book for us to write. A combined effort of a few people with varied backgrounds—spanning information security, software development, and online virtual lab development and teaching—this book should appeal and relate to many people.

Wireshark is *the* tool for capturing and analyzing network traffic. Originally named Ethereal but changed in 2006, Wireshark is well established and respected among your peers. But you already knew that, or why would you invest your time and money in this book? What you’re really here for is to delve into how Wireshark makes your job easier and your skills more effective.

**Overview of the Book and Technology**

This book hopes to meet three goals:

- Broaden the information security professional’s skillset through Wireshark.
- Provide learning resources, including labs and exercises, to apply what you learn.
- Demonstrate how Wireshark helps with real-life scenarios through Lua scripting.

The book isn’t only for reading; it’s for doing. Any Wireshark book can show how wonderful Wireshark can be, but this book also gives you opportunities to practice the craft, hone your skills, and master the features Wireshark offers.

These opportunities come in a few forms. First, to apply what’s in the text, you will practice in labs. You build the lab environment early on the book and put it to use throughout the chapters that follow. The second opportunity
for practice is at the end of each chapter, save the last Lua scripting chapter. The end-of-chapter exercises largely build on the labs to challenge you again, but with far less hand-holding. Between the labs and exercises, your time spent with Wireshark ensures time spent reading is not forgotten.

The lab environment was created using containerization technology, resulting in a fairly lightweight virtual environment to be installed and run on your own system. The whole environment was designed specifically for you, the book reader, to practice the book's content. These labs were developed and are maintained by one of the authors, Jessey Bullock. The source code for the labs is available online. See Chapter 2 for specifics.

In short, this book is a hands-on, practice-oriented Wireshark guide created for you, the information security professional. The exercises will help you to keep you advancing your Wireshark expertise long after the last page.

How This Book Is Organized

The book is structured on the assumption that readers will start from the beginning and then work through the main content. The initial three chapters not only introduce the title application Wireshark but also the technology to be used for the labs, along with the basic concepts required of the reader. Readers already familiar with Wireshark should still work through the lab setup chapter, since future chapters depend on the work being done. These first three chapters are necessary to cover first, before putting the following chapters to use.

The majority of the book that follows is structured to discuss Wireshark in the context of information security. Whether capturing, analyzing, or confirming attacks, the book's main content and its labs are designed to most benefit information security professionals.

The final chapter is built around the scripting language Lua. Lua greatly increases Wireshark's flexability as an already powerful network analyzer. Initially, the Lua scripts were scattered throughout chapters, but they were later combined into a single chapter all their own. It was also appreciated that not all readers are coders, so Lua scripts are better served through one go-to resource.

Here's a summary of the book's contents:

Chapter 1, “Introducing Wireshark,” is best for the professional with little to no experience with Wireshark. The main goal is to help you avoid being overwhelmed, introduce the interface, and show how Wireshark can be your friend.

Chapter 2, “Setting Up the Lab,” is not to be skipped. Starting with setting up a virtualized machine, this chapter then sets up the W4SP Lab, which you will use several times in upcoming chapters.

Chapter 3, “The Fundamentals,” covers basic concepts and is divided into three parts: networking, information security, and packet analysis. The book assumes most readers might be familiar with at least one or two areas, but the chapter makes no assumptions.
Chapter 4, “Capturing Packets,” discusses network captures, or the recording of network packets. We take a deep dive into how Wireshark captures, manipulates capture files, and interprets the packets. There’s also a discussion around working with the variety of devices you encounter on a network.

Chapter 5, “Diagnosing Attacks,” makes good use of the W4SP Lab, re-creating various attacks commonly seen in the real world. Man in the middle attacks, spoofing various services, denial of service attacks and more are all discussed.

Chapter 6, “Offensive Wireshark,” also covers malicious traffic, but from the hacker’s perspective. Wireshark and the W4SP Lab are again relied on to launch, debug, and understand exploits.

Chapter 7, “Decrypting TLS, Capturing USB, Keyloggers, and Network Graphing,” is a mash-up of more activities as we leverage Wireshark. From decrypting SSL/TLS traffic to capturing USB traffic across multiple platforms, this chapter promises to demonstrate something you can use wherever you work or play.

Chapter 8, “Scripting with Lua,” contains about 95% of the book’s script content. It starts simple with scripting concepts and Lua setup, whether you’re working on Windows or Linux. Scripts start with “Hello, World” but lead to packet counting and far more complex topics. Your scripts will both enhance the Wireshark graphic interface and run from the command line.

Who Should Read This Book

To claim this book is for security professionals might be specific enough to the general IT crowd. However, to most information security professionals, it’s still too broad a category. Most of us specialize in some way or another, and identify ourselves by our role or current passion. Some examples include firewall administrator, network security engineer, malware analyst, and incident responder.

Wireshark is not limited to just one or two of those roles. The need for Wireshark can be found in roles such as penetration tester or ethical hacker—roles defined by being proactive and engaging. Additional roles like forensics analyst, vulnerability tester, and developer also benefit from being familiar with Wireshark. We’ll show this through examples in the book.

Regarding expectations on the reader, the book makes no assumptions. Information security specializations vary enough so that someone with 15 years of experience in one field is likely a novice in other fields. Wireshark offers value for anyone in those fields, but it does expect a basic understanding of networking, security and how protocols work. Chapter 3 ensures we’re all on the same page.

Any reader must be technically savvy enough to install software or understand systems are networked. And since the book targets security professionals, we presume a fundamental level for information security. Still, as far as
“fundamentals” go, Chapter 3 acts as a refresher for what’s necessary around networking, information security, and packet and protocol analysis.

Further in the book, Wireshark is used in the context of various roles, but there’s no experience requirement for grasping the content or making use of the labs. For example, the tools used in Chapter 6, “Offensive Wireshark” might be already familiar to the penetration tester, but the chapter assumes zero experience when instructing setup.

To sum up, we understand there is a wide spectrum of possible roles and experience levels. You might be employed in one of these roles and want to use Wireshark more. Or you might be getting ready to take on one of these roles, and recognize Wireshark as essential tool to use. In either case, this book is for you.

Tools You Will Need

The one tool required for this book is a system. Your system does not need to be especially powerful; at the most a few years old would be best. Your system will be first used in Chapter 2, “Setting Up the Lab.” You first install and set up a virtualized machine. Then upon that virtual machine you will set up the labs.

Of course, this book can benefit those without a system, but a system is needed to perform the labs referenced throughout the book.

What’s on the Website

The primary website needed for this book is the GitHub repository for the W4SP Lab code. The GitHub repo and its contents are explained further in Chapter 2, “Setting Up the Lab,” where you first download and build the virtual lab environment. Then the Lab files are installed onto your virtual machine.

Other websites are cited throughout the book, mostly as pointers for additional resources. For example, some sites hold hundreds of network capture files that are available for analysis.

Summary

This is where the authors are at the edge of our seats, hoping you will leap into and enjoy the book, its materials, and the labs. A lot of thought and effort went into this book. Our only desire was to create a resource that inspired more people to have a deeper appreciation of Wireshark. Being information security professionals ourselves, we crafted this book for our peers.
Welcome to *Wireshark for Security Professionals*. This introductory chapter covers three broad topics. In the first part, we discuss what Wireshark is used for and when to use it.

The second part of this chapter introduces the popular graphic user interface (GUI). The GUI for Wireshark can appear quite busy at first, so we immediately want to get familiar with its layout. We break down the different areas of the interface, how they relate to one another, and the reasoning for needing each one. We also discuss how and when each part of the interface helps you maximize your use of Wireshark.

In the third part of this chapter, we discuss the way Wireshark filters data presented on the interface. Being familiar with Wireshark’s interface helps you appreciate all the data presented, but the amount of data can still be overpowering. Wireshark offers ways to filter or separate what you need from all that is presented. The last part is about different types of filters and how you can customize these filters.

Wireshark can appear to be a complicated tool, but by the end of this first chapter, the hope is you have a much higher comfort level with the tool’s purpose, interface, and ability to present you with what you want to see.
What Is Wireshark?

Wireshark, in its most basic sense, is a tool to understand data you capture from a network. The captured data is interpreted and presented in individual packet form for analysis, all within Wireshark. As you probably already know, packets are the chunks of data streaming on a network. (Technically, depending on the context level of where in the system the data is interpreted, chunks are called frames, datagrams, packets, or segments, but we’ll just use “packets” for now.) Wireshark is a network and protocol analyzer tool, free for download and use on a variety of platforms, spanning many flavors of Unix and Windows.

Wireshark first captures the data from a network interface and then breaks the capture into the frames, segments, and packets, understanding where they begin and end. Wireshark then interprets and presents this data in the context of addressing, protocols and data. You can analyze the captures immediately or save them to load later and share with others. In order for Wireshark to view and capture all packets, not just those involving the capturing system, the network interface is placed in promiscuous mode (also called monitor mode) in the context of capturing on a wireless network. Finally, what grants you the ability to analyze packets in Wireshark are the dissectors. All these basic elements are discussed in more detail in Chapter 4, in the context of “sniffing” or capturing data, and how that captured data is interpreted.

A Best Time to Use Wireshark?

Wireshark is an immensely powerful tool with quite a bit of deep and complex functionality. It is capable of handling a wide range of known (and unknown) protocols. But although the functionality range is broad, most of it aligns to one end: to capture packets and analyze them. Being able to take the bits and bytes and present them in an organized, familiar, and human-readable format is what brings people to think of using Wireshark.

Before launching Wireshark, it’s important to understand when to use it and when not to use it. Sure, it’s a great tool, but like any tool, it’s best used when it’s the right tool for the job.

Here are scenarios when it’s ideal to use Wireshark:

- To look for the root cause of a known problem
- To search for a certain protocol or stream between devices
- To analyze specific timing, protocol flags, or bits on the wire

And while not ideal, Wireshark can also be used:

- To discover which devices or protocols are the top talkers
- To see a rough picture of network traffic
- To follow a conversation between two devices
You get the idea. Wireshark is ideal for determining a root cause of an understood problem. While not ideal for browsing network traffic or making high-level judgments about the network, Wireshark does have some features to show those statistics. But Wireshark can’t and shouldn’t be the first tool thought of early on in discovering a problem. Someone who opens Wireshark to skim through the list of packets to assess network health would soon be overwhelmed. Instead, Wireshark is for problem solvers, for the detectives who already know their suspects well.

**Avoiding Being Overwhelmed**

The majority of people who walk away from Wireshark do so because they find it overwhelming after only a few early experiences. To label Wireshark as overwhelming is misleading, however. What really paralyzes new users is the traffic, the list of packets flying by, not the application’s functionality. And, fair enough, once you start a capture and the packets scroll by in real time, it’s definitely intimidating. (But that’s what filters are for!)

To avoid being overwhelmed, consider two aspects of Wireshark before diving into it:

- **The interface**—how it’s laid out and why
- **Filters**—how they work to reveal what you want

Once you get a quick appreciation of the tool’s interface and how to write a filter, Wireshark suddenly appears intuitive and shows its power, without the scare factor. And that’s what we focus on for the rest of this chapter.

The following sections are on the most important aspects that you need immediately to be comfortable using Wireshark. If you are already familiar with Wireshark, as well as filters, feel free to skim this chapter as a refresher so that you can be sure you are on the same page for the rest of the book.

**The Wireshark User Interface**

We start with the busy Wireshark GUI, which is packed with features. We provide a high-level overview of where you need to look to start seeing some packet data. With packet capturing covered, we then discuss the more powerful features of Wireshark, starting with dissectors. In Wireshark, dissectors are what parse a protocol and decode it for presenting on the interface. They enable Wireshark to give the raw bits and bytes streaming across the wire some context by displaying them into something more meaningful to the human analyst. We then round off the chapter by covering the various filters available to help limit and zero in on just the network data you are interested in.

The home screen appears when you open Wireshark. On this screen are shortcuts you can use to start a new capture or open a previous capture file.
For most newcomers to Wireshark, the brightly colored Capture button is the most attractive option. Starting a capture leads to a flurry of scrolling packets, which for the newcomer then leads to overwhelm. But let’s go back to the home screen. There are also links to online documentation that you can use to figure out how to accomplish a certain task.

On the top of the screen, as shown in Figure 1-1, is the menu bar in the classic format you are probably familiar with. These menus have settings and other features like statistics that can be accessed when needed. (Don’t worry—we aren’t really worried about statistics.) Below these menus is the Main toolbar, which has quick access icons for the functionality you will use most while analyzing network traffic. These icons include things like starting or stopping a capture, and the various navigation buttons for finding your way around captured packets. Icon buttons are typically grayed if not applicable or usable—for example, without a capture yet.

Icons change over time from version to version. At the time this book was written, the blue shark fin starts a capture and the red square stops a capture. The shark fin is gray until the network interface is chosen, and we cover that soon. Also note that this toolbar area gives you a visual indication of the capture process. Again, many options are grayed out in Figure 1-1 because we are not yet capturing or don’t have a capture completed. As you go through this chapter, pay attention to this area to understand how it changes and how it reflects the various capture states. In many respects, Wireshark has an intuitive user experience.

Figure 1-1: The Wireshark home screen
The Filter toolbar, which is below the Main toolbar, is a vital part of the Wireshark UI. You will soon fall in love with this little box, as you often find yourself drowning in a torrent of traffic. The Filter toolbar lets you remove whatever is uninteresting to the task at hand and presents just what you’re looking for (or takes out what you’re not looking for). You can enter display filters in the Filter text box that help you drill down what packets you see in the Packet List pane. We discuss filters in detail later in this chapter, but for now just trust me: They will be your new best friends.

Packet List Pane

The largest portion in the middle of the interface is reserved for the packet list. This list shows all the packets captured along with useful information, such as source and destination IP, and the time difference between when the packets were received. Wireshark supports color coding various packets to make sorting of traffic and troubleshooting easier. You can add custom colors for packets of interest, and the columns within the Packet List pane display useful information such as the protocol, packet length, and other protocol-specific information (see Figure 1-2).

Figure 1-2: The Packet List pane

This window is the bird’s-eye view into the network you are sniffing or the packet capture you have loaded into Wireshark. The last column, by default
labeled “Info,” offers a quick summary of what that packet contains. Of course, it depends on the packet, but it might be the URL for an HTTP request or the contents of a DNS query, which is really useful for getting a quick handle on important traffic in your capture.

**Packet Details Pane**

Below the Packet List pane is the Packet Details pane. The Packet Details pane shows information for the selected packet in the Packet List pane. This pane contains a ton of information, down to what the various bytes are within the packet. Information such as the source and destination MAC address is included here. The next row contains IP information. The next row reveals the packet is sending to UDP port 58351. The next row reveals what information is contained in that UDP packet.

These rows are ordered by the headers as they are ordered when sending data on the network. That means they are subject to change if you are capturing on a different type of network, such as a wireless network, that has different headers. The DNS column, which is the application data encapsulated within UDP, is expanded in Figure 1-3. Notice how Wireshark allows you to easily pull out information, such as the actual DNS query that was made within this DNS packet. This is what makes Wireshark the powerful network analysis tool that it is. You don’t have to memorize the DNS protocol to know which bits and bytes at what offset translate into a DNS query.

![Figure 1-3: The Packet Details pane](image-url)
Subtrees

Because the details would be overwhelming if shown all at once, the information is organized and collapsed into sections. The sections, called subtrees, can be collapsed and expanded to display only what you need. (In Figure 1-2, the subtrees are collapsed; in Figure 1-3, they are expanded.)

**NOTE** You might hear the message sent between devices referred to as a data frame or a packet. But what’s the difference? When referring to the message at the OSI layer 2 (the data link layer, where the MAC address is used), the whole message is called a frame. When referring to the message at OSI model layer 3 (the network layer, for example, using the IP address), then the message is called a packet.

If you’re already familiar with how a data frame is structured, you recognize how the packet details subtrees are divided. Details are structured into subtrees along the lines of the data frame’s headers. You can collapse/expand a subtree by clicking the arrow sign next to the relevant section. The arrow is pointing to the right if the subtree is collapsed. Once you click on the arrow to expand that subtree, you’ll see the arrow points down (refer to Figure 1-3). And, of course, you’ll always have the option to expand or collapse all subtrees by right-clicking anywhere in the Packet Details pane to launch its pop-up menu.

In Figures 1-2 and 1-3, packet number 7 is selected. Whatever packet is selected in the Packet List pane is the packet presented in the panes below it. In this case, it’s packet number 7 showing within the Packet Details pane.

**NOTE** Packets are usually numbered based on the time they are received, although this isn’t guaranteed. The packet capture (pcap) library determines how to order the packets.

If you double-click this packet, a separate window appears, to open the packet details. This is useful when you want to visually compare two different packets quickly. The Packet Details area in Figure 1-3 shows various rows of information that can be expanded or collapsed.

Capturing Enough Detail

The first row contains metadata regarding the packet, such as the number of the packet, when it was captured, on what interface it was captured, and the number of bytes captured versus the number of bytes that were on the wire. That last part might sound a little strange. Wouldn’t you always capture all the bytes that go across the wire? Not necessarily. Some network capture tools allow you to capture only a subset of the bytes that are actually transmitted across the wire. This is useful if you only want to get an idea of the type of packets that are going across the wire but not what actual data those packets
have, which can greatly reduce the size of the packet capture. The downside, of course, is that you get only a limited amount of information. If disk space is not an issue, feel free to capture it all. Just be mindful that you are capturing and storing all traffic traversing that network cable, which can quickly become a significant amount.

There are ways to limit the size of the capture. For example, instead of truncated packet data, capture only specific packet types and not all traffic. If someone wants to send you a capture, or if you want to see specific traffic, you can have Wireshark capture only the traffic you want, saving space. Everything is done using the right filters—and that section is coming soon enough!

**Packet Bytes Pane**

What follows the Packet Details pane is the Packet Bytes pane. This pane is at the bottom of the screen and wins the award for least intuitive. At first glance, it simply looks like gibberish. Bear with me for a couple of paragraphs; it will all make sense soon.

*Offsets, Hex, and ASCII*

You can see the Packet Bytes pane is divided into three columns. The first, left-most column simply counts incrementally: 0000, 0010, 0020, and so on. That’s the offset (in hexadecimal) of the selected packet. Here, *offset* simply means the number of bits off from the beginning—again, counting in hexadecimal (where 0x0010 = 16 in decimal). The middle column shows information, in hexadecimal, at that offset. The right-hand column shows the same information, but in ASCII. For example, the total amount of information from the very beginning (offset 0000) to offset 0010 is 16 bytes. The middle column shows each of the 16 bytes in hex. The right-hand column shows each of the 16 bytes in ASCII characters. When a hexadecimal value doesn’t translate to a printable ASCII character, only a “.” (period), is shown. So the Packet Bytes pane is actually the raw packet data as seen by Wireshark. By default, it is displayed in hex bytes.

Right-clicking the pane gives you the option to convert the hex bytes into bits, which is the purest representation of the data, though often this might not be as intuitive as the hex representation. Another neat feature is that any row you highlight within the Packet Details pane causes the corresponding data within the Packet Bytes pane to be highlighted. This can be helpful when troubleshooting Wireshark’s dissection, as it allows you to see exactly which packet bytes the dissector is looking at.
Filters

When you start your first packet capture, a lot will probably be going on in the Packet List pane. The packets move across the screen too fast to make sense of anything meaningful. Fortunately, this is where filters can help. Filters are the best way to quickly drill down to the information that matters most during your analysis sessions. The filtering engine in Wireshark allows you to narrow down the packets in the packet list so that communication flows or certain activity by network devices becomes immediately apparent.

Wireshark supports two kinds of filters: display filters and capture filters. Display filter are concerned only with what you see in the packet list; capture filters operate on the capture and drop packets that do not match the rules supplied. Note that the syntax of the two types of filters is not the same.

Capture filters use a low-level syntax called the Berkeley Packet Filter (BPF), whereas display filters use a logic syntax you will recognize from most popular programming languages. Three other packet-capturing tools—TShark, Dumpcap, and tcpdump—also use BPF for capture filtering, as it’s quick and efficient. TShark and Dumpcap are both command-line packet-capturing tools and provide analysis capabilities, the former being the command-line counterpart to Wireshark. TShark, covered more deeply with example output, is introduced in Chapter 4. The third, tcpdump, is strictly a packet-capturing tool.

Generally, you use capture filters when you want to limit the amount of network data that goes into processing and is getting saved; you use display filters to drill down into only the packets you want to analyze once the data has been processed.

Capture Filters

There are times when capturing network traffic that you can limit the traffic you want beforehand; at other times you will have to because the capture files will grow too large too fast if you don’t start filtering. Wireshark allows you to filter traffic in the capture phase. This is somewhat similar to the display filters, which you will read about later in this chapter, but there are fewer fields that can be used to filter on, and the syntax is different. It’s most important to understand that a capture filter screens packets before they are captured. A display filter, however, screens what saved packets are displayed. Therefore, a restrictive capture filter means your capture file will be small (and thus a smaller number of displayed packets, too). But using no capture filter means
capturing every packet, and thus a large capture file, on which display filters can be used to narrow the list of packets shown.

While it makes sense for Wireshark to capture everything by default, it does actually use default capture filters in some scenarios. If you are using Wireshark on a remote session, such as through Remote Desktop or through SSH, then capturing every packet would include many packets relaying the session traffic. Upon startup, Wireshark checks to see whether a remote session is in use. If so, a capture filter to filter out remote session traffic is in use by default.

The building blocks of a capture filter are the protocol, direction, and type. For example, tcp dst port 22 captures only TCP packets with a destination port of 22. The possible types are:

- host
- port
- net
- portrange

Direction can be set using src or dst. As you suspect, src is for capturing from a specified source address, while dst can specify the destination address. If it is not specified, both will be matched. In addition to specifying one direction, the following combined direction modifiers can be used: src or dst and src and dst.

In a similar way, if a type is not specified, a host type will be assumed. Note that you need to specify at least one object to compare to; the host modifier will not be assumed if you would only specify an IP address as filter and will result in a syntax error.

The direction and protocol can be omitted to match a type in both source and destination across all protocols. For example, dst host 192.168.1.1 would only show traffic going to the specified IP. If dst is omitted, it would show traffic to and from that IP address.

The following are the most commonly used BPF protocols:

- ether (filtering Ethernet protocols)
- tcp (filtering TCP traffic)
- ip (filtering IP traffic)
- ip6 (filtering IPv6 traffic)
- arp (filtering ARP traffic)

In addition to the standard components, there is a set of primitives that do not fit in one of the categories:

- gateway (matches if a packet used the specified host as gateway)
- broadcast (for broadcast, not unicast, traffic)
- less (less than, followed by a length)
- greater (greater than, followed by a length)

These primitives can be combined with the other components. For example, ether broadcast will match all Ethernet broadcast traffic.

Capture filter expressions can be strung together using logical operators. Again, with both the English and the logical notation:

- and (&&)
- or (||)
- not (!)

For example, here are some filters for systems named alpha and beta:

- host beta (captures all packets to and from the alpha system)
- ip6 host alpha and not beta (captures all IP packets between alpha and any host except beta)
- tcp port 80 (captures all TCP traffic across port 80)

**Debugging Capture Filters**

Capture filters operate on a low level of the captured network data. They are compiled to processor opcodes (processor language) in order to ensure high performance. The compiled BPF can be shown by using the -d operator on tcpdump, Dumpcap, or TShark, and in the Capture Options menu in the GUI. This is useful when debugging a problem where your filter is not doing exactly what you were expecting. The following is an example output of a BPF filter:

```
localhost:~$ dumpcap -f "ether host 00:01:02:03:04:05" -d
Capturing on 'eth0'
(000) ld       [8]
(001) jeq      #0x2030405       jt 2    jf 4
(002) ldh      [6]
(003) jeq      #0x1             jt 8    jf 4
(004) ld       [2]
(005) jeq      #0x2030405       jt 6    jf 9
(006) ldh      [0]
(007) jeq      #0x1             jt 8    jf 9
(008) ret      #65535
(009) ret      #0
```

As previously mentioned, using the -d operator will show the BPF code for the capture filter. And, used in the example above, the -f operator will show the libpcap filter syntax.