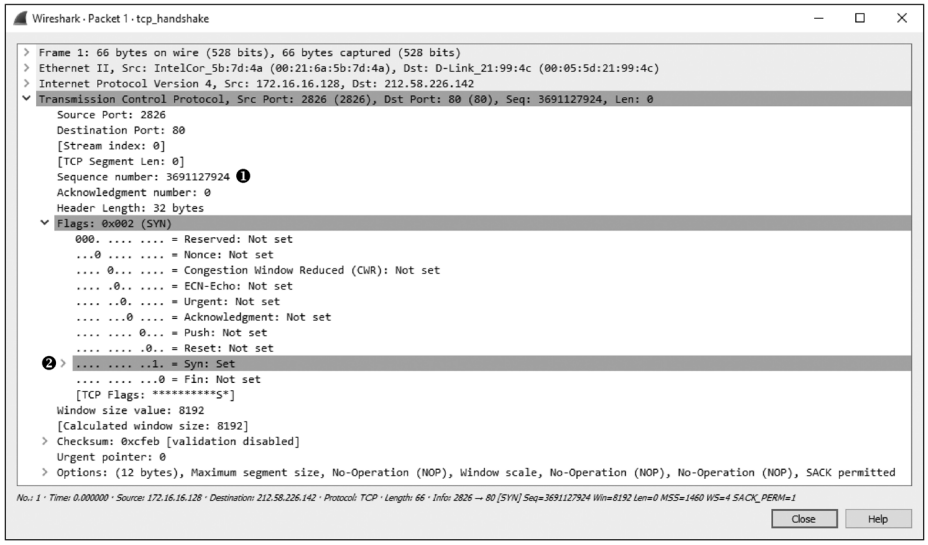


图 8-6　*初始 SYN 数据包*

握手中第二个数据包是从 212.58.226.142 发出的 SYN/ACK 响应（见图 8-7）。这个数据包也包含着这台主机的初始序列号（233779340）以及一个确认号（2691127925）。这个确认号比之前的那个数据包序列号大 1，因为这个域是用来表示主机所期望得到的下一个序列号的值的。

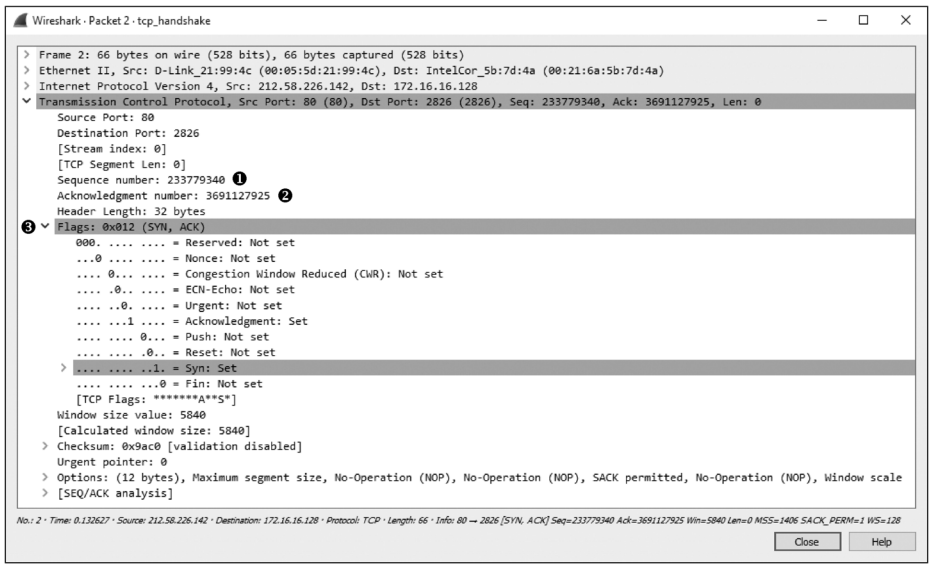


图 8-7　SYN/ACK *响应*

异步社区Kensuke(18319066421) 专享 请尊重版权

最后的数据包是从 172.16.16.128（见图 8-8）发出的 ACK 数据包。这个数据包正如所期望的那样，包含着之前数据包确认号域所定义的序列号 3691127925。

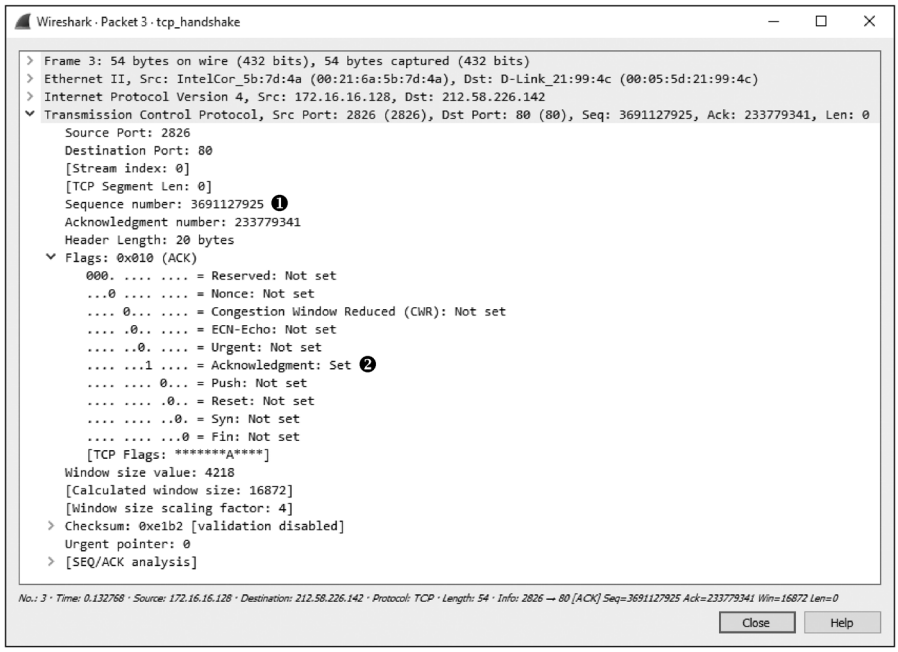


图 8-8　最后的 ACK 包

握手发生在每个 TCP 的通信序列之前。当在一个繁忙的捕获文件中搜索通信序列的开头时，序列 SYN、SYN/ACK、ACK 是一个很好的标志。

8.1.4　TCP 链接断开

所有的问候最终都会有一句再见，在 TCP 中，每一个握手和终止使得连接有始有终。TCP 终止用来在两台设备完成通信后正常地结束连接。这个过程包含 4 个数据包，并且用一个 FIN 标志来表明连接的终结。

在一个终止序列中，主机 A 通过发送一个设置了 FIN 和 ACK 标志的 TCP 数据包，告诉主机 B 通信完成。主机 B 以一个 ACK 数据包响应，并传输自己的 FIN/ACK 数据包。主机 A 响应一个 ACK 数据包，然后结束通信过程。这个过程如图 8-9 所示。

异步社区Kensuke(18319066421) 专享 请尊重版权

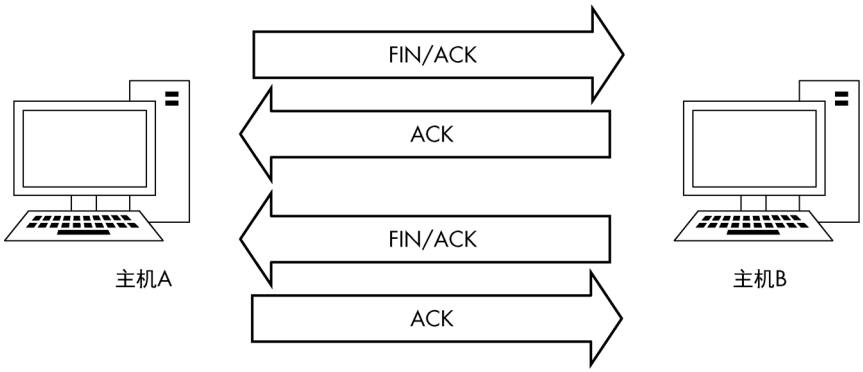


图 8-9　TCP 终止过程

打开文件 tcp_teardown.pcapng 可以在 Wireshark 中看到这个过程。在序列的第一个数据包（见图 8-10），你可以看到位于 67.228.110.120 的设备通过发送有着 FIN 和 ACK 标志的数据包，来开始终止过程。

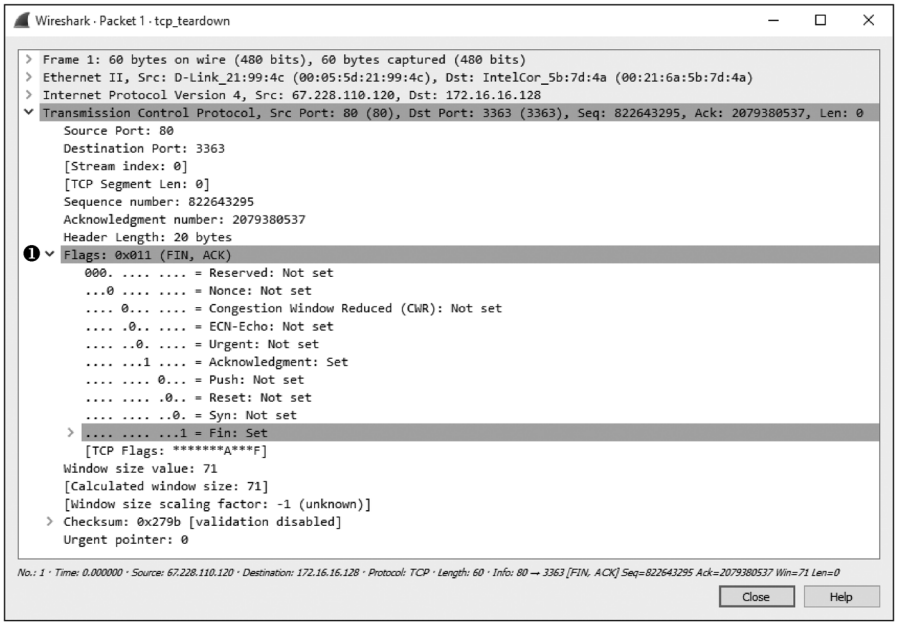


图 8-10　FIN/ACK 包作为终止过程的开始

在这个数据包被发出去之后，172.16.16.128 使用了一个 ACK 数据包进行响应，来确认第一个数据包的接收，然后发送了一个 FIN/ACK 数据包。整个过程在 67.228.110.120 发送了最终的 ACK 之后结束。这时，这两个设备的通信便已

异步社区Kensuke(18319066421) 专享 请尊重版权

经结束，如果想要再次开始通信就必须完成新的 TCP 握手。

8.1.5 TCP 重置

在理想情况下，每一个连接都会以 TCP 终止来正常结束。但在现实中，连接经常会突然断掉。举例来说，这可能是由于一个潜在的攻击者正在进行端口扫描，或者仅仅是主机配置错误所导致。在这些情况下，就需要使用设置了 RST 标志的 TCP 数据包。RST 标志用来指出连接被异常中止，或拒绝连接请求。

文件 tcp_refuseconnection.pcapng 给出了一个包含有 RST 数据包网络流量的例子。这个文件中的第一个数据包来自 192.168.100.138，其尝试与 192.168.100.1 的 80 端口进行通信。这个主机不知道 192.168.100.1 并没有在监听 80 端口，因为那是一个思科路由器，并且并没有配置 Web 接口，也就是说并没有服务监听 80 端口的连接。为了响应这个连接请求，192.168.100.1 向 192.168.100.138 发送了一个数据包，告诉它其对于 80 端口的通信无效。图 8-11 中展示了在第二个数据包的 TCP 头中这个连接尝试突然终止的情况。RST 数据包除了包含 RST 和 ACK 标志外，没有任何其他的东西，之后也并没有额外的通信。

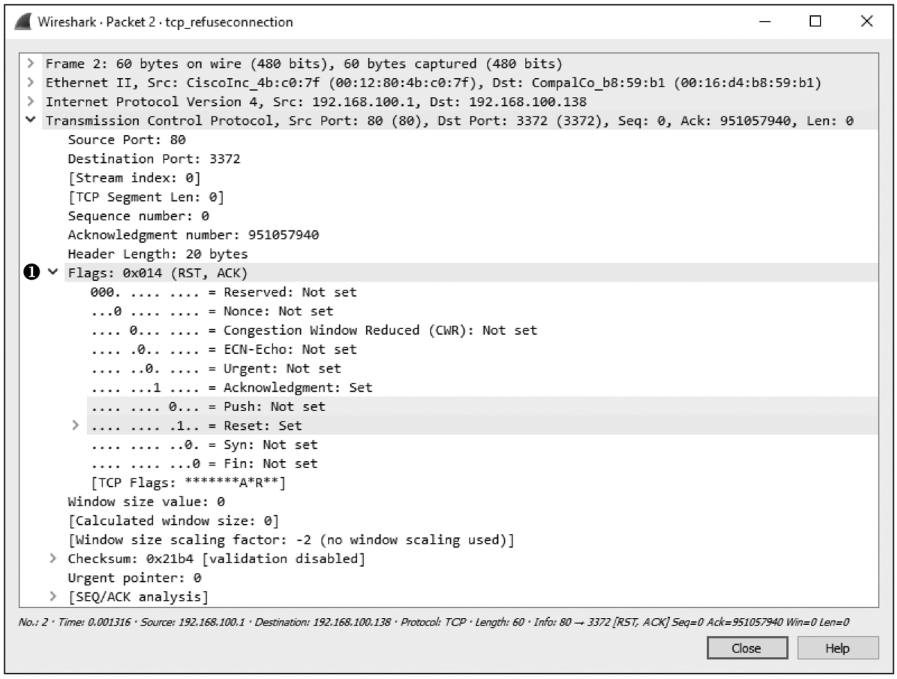


图 8-11　RST 和 ACK 标志代表着通信的结束

异步社区Kensuke(18319066421) 专享 请尊重版权

RST 数据包会在尝试通信序列的开始（就像这个例子一样）或者在主机通信的中途，来终止通信。

8.2 用户数据报协议

用户数据报协议（User Datagram Protocol, UDP）是在现代网络中较常使用的另外一种第 4 层协议。如果说 TCP 是为了满足带有内在错误检测的可靠数据传输，那么 UDP 主要是为了提供高速的传输。出于这个原因，UDP 是一种尽力服务，通常会被称为无连接协议。一个无连接协议并不会正式地建立和结束主机之间的连接，也不会像 TCP 那样存在握手和终止过程。

无连接协议也就意味着不提供可靠性服务，这么看上去 UDP 流量一点都不稳定。事实上也正是如此，但依赖于 UDP 的协议通常都会有其自己内置的可靠性服务，或者使用 ICMP 的一些功能，来保证连接更可靠一些。举例来说，应用层协议 DNS 和 DHCP 高度依赖于数据包在网络上传输的速度，虽然使用了 UDP 作为它们的传输层协议，但还是它们自己进行错误检查以及重传计时。

UDP 报头结构

UDP 头比 TCP 头要小得多，也简单得多。如图 8-12 所示，以下是 UDP 报头的字段。

源端口：用来传输数据包的端口。

目标端口：数据包将要被传输到的端口。

数据包长度：数据包的字节长度。

校验和：用来确保 UDP 头和数据到达时的完整性。

用户数据报协议 (UDP)					
偏移位	八位组	0	1	2	3
八位组	位	0–7	8–15	16–23	24–31
0	0	源端口		目标端口	
4	32	数据包长度		校验和	

图 8-12 UDP 报头

异步社区Kensuke(18319066421) 专享 请尊重版权

文件 udp_dnsrequest.pcapng 中包含有一个数据包，这个数据包是一个使用 UDP 的 DNS 请求。当展开这个数据包的 UDP 头时，你可以看到 4 个域（见图 8-13）。

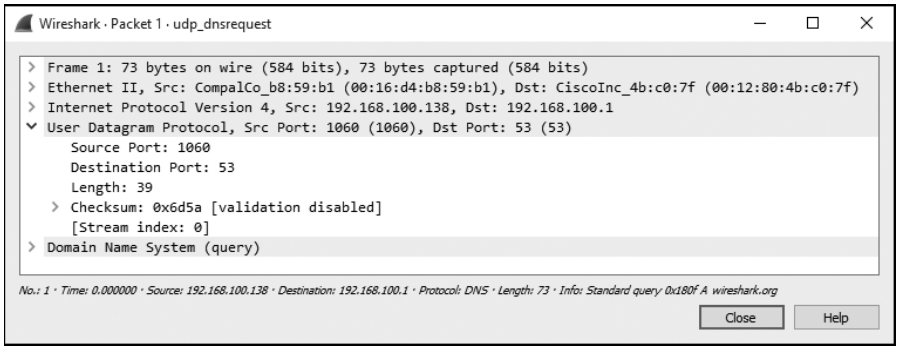


图 8-13　UDP 数据包的内容非常简单

需要记住的是，UDP 并不关心传输的可靠性，所以任何使用 UDP 的应用在必要的时候都需要采取特殊的步骤，保证可靠的传输。这一点和 TCP 相反，TCP 有自己的一套连接正式发起和结束的程序，也有自己的一些机制来校验数据包的成功传输。

这一章向你介绍了传输层协议——TCP 和 UDP。不像网络层协议，TCP 和 UDP 是日常在绝大多数网络上交互的核心，因此能否掌握有效分析它们的能力将会决定你能否成为一位数据包分析大师。在第 9 章，我们将看一看常见的应用层协议。

异步社区Kensuke(18319066421) 专享 请尊重版权

第9章

常见高层网络协议



在这一章中，我们将继续介绍一些协议的功能，以及它们在 Wireshark 中的样子。我们将介绍 4 种常见的高层协议（第 7 层）：DHCP、DNS、HTTP 和 SMTP。

9.1 动态主机配置协议 DHCP

在网络时代的早期，当一台设备想要在网络上通信时，它需要被手动分配一个地址。随着网络的发展，这样的手动过程很快就变得烦琐起来。为了解决这个问题，BOOTP 协议（Bootstrap Protocol）问世，它主要的作用是给连接到

异步社区Kensuke(18319066421) 专享 请尊重版权

网络的设备自动分配地址。BOOTP 后来被更加复杂的动态主机配置协议 DHCP（Dynamic Host Configuration Protocol）所取代。

DHCP 是一个应用层协议，负责让设备能够自动获取 IP 地址（以及其他重要的网络资源，比如 DNS 服务器和路由网关的地址）。今天大多数的 DHCP 服务器都向客户端提供其他的一些参数，比如网络上的默认网关和 DNS 服务器的地址。

9.1.1 DHCP 头结构

DHCP 数据包会为客户端带来很多信息。如图 9-1 所示，以下字段都会在 DHCP 数据包中出现。

动态主机配置协议(DHCP)					
偏移位	八位组	0	1	2	3
八位组	位	0–7	8–15	16–23	24–31
0	0	操作代码	硬件类型	硬件长度	跳数
4	32	事务ID			
8	64	消耗时间		标志	
12	96	客户端IP地址			
16	128	“你的”IP地址			
20	160	服务器IP地址			
24	192	网关IP地址			
28	224	客户端IP地址			
32	256	客户端硬件地址(16 bytes)			
36	288				
40	320				
44	352				
48+	384+	服务器主机名 (64 bytes)			
		启动文件(128 bytes)			
		选项			

图 9-1　DHCP 数据包结构

操作码（OpCode）：用来指出这个数据包是 DHCP 请求还是 DHCP 回复。

硬件类型（Hardware Type）：硬件地址类型（10MB 以太网、IEEE802、ATM 以及其他）。

硬件地址长度（Hardware Length）：硬件地址的长度。

跳数（Hops）：中继代理用以帮助寻找 DHCP 服务器。

异步社区Kensuke(18319066421) 专享 请尊重版权

事务 ID（Transaction ID）：用来匹配请求和响应的一个随机数。

消耗时间（Seconds Elapsed）：从客户端第一次向 DHCP 服务器发出地址请求到获得响应所需要的时间。

标志（Flags）：DHCP 客户端能够接收的流量类型（单播、广播以及其他）。

客户端 IP 地址（Client IP Address）：客户端的 IP 地址（由“你的”IP 地址域派生）。

“你的”IP 地址（Your IP Address）：DHCP 服务器提供的 IP 地址（最终成为客户端 IP 地址域的值）。

服务器 IP 地址（Server IP Address）：DHCP 服务器的 IP 地址。

网关 IP 地址（Gateway IP Address）：网络默认网关的 IP 地址。

客户端硬件地址（Client Hardware Address）：客户端的 MAC 地址。

服务器主机名（Server Host Name）：服务器的主机名（可选）。

启动文件（Boot File）：DHCP 所使用的启动文件（可选）。

选项（Options）：用来对 DHCP 数据包进行扩展，以提供更多功能。

9.1.2 DHCP 续租过程

DHCP 的主要任务就是在续租过程中向客户端分配 IP 地址。续租过程在一个客户端和 DHCP 服务器之间进行，如文件 dhcp_nolease_renewal.pcap 中所示。DHCP 的续租过程通常被称为 DORA 过程，因为它使用了 4 种类型的 DHCP 数据包：发现（Discover）、提供（Offer）、请求（Request）和确认（Acknowledgement），如图 9-2 所示。在这里，我们将对 DORA 数据包的每种类型进行逐一介绍。

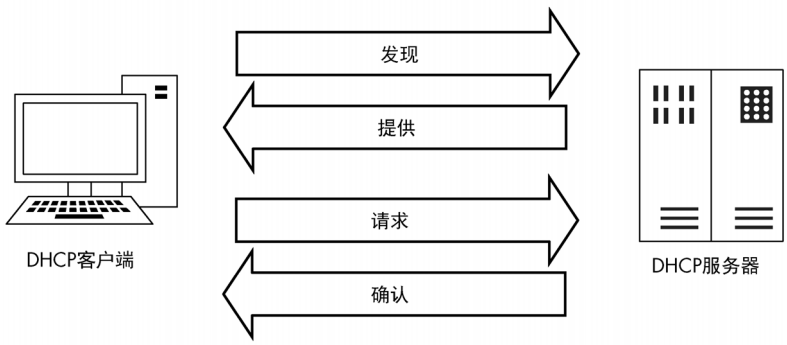


图 9-2　DHCP 的 DORA 过程

异步社区Kensuke(18319066421) 专享 请尊重版权

1. 发现数据包

正如在引用的捕获文件中看到的那样，第一个数据包从 0.0.0.0 的 68 端口发往 255.255.255.255 的 67 端口。客户端使用 0.0.0.0，是因为它目前还没有 IP 地址。数据包被发往 255.255.255.255，是因为这是一个独立于网络的广播地址，从而确保这个数据包会被发往网络上的每台设备。因为这台设备并不知道 DHCP 服务器的地址，所以它的第一个数据包是为了寻找正在监听的 DHCP 服务器。

在 Packet Details 面板中，我们第一眼就可以看到 DHCP 是基于 UDP 作为传输层协议的。DHCP 对于客户端得到其所请求信息的速度有很高的要求。由于 DHCP 有其内置的保证可靠性的方法，因此也就意味着 UDP 是比较适合的协议。你可以在第一个数据包的 Packet Details 面板的 DHCP 部分，查看发现过程的细节，如图 9-3 所示。

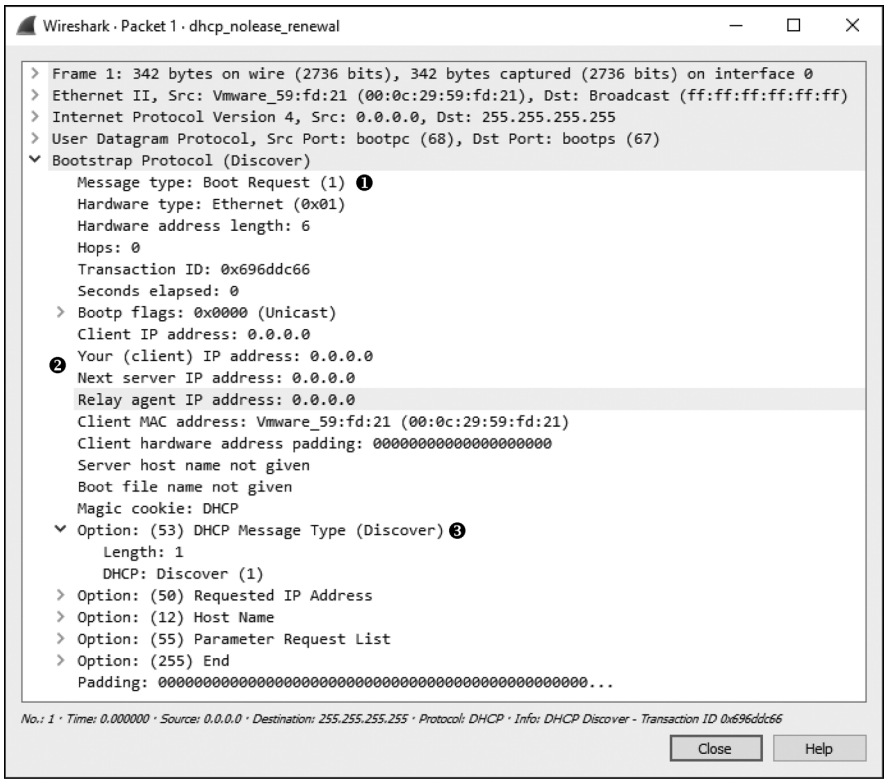


图 9-3　DHCP 发现数据包

异步社区Kensuke(18319066421) 专享 请尊重版权

注意　由于 Wireshark 在处理 DHCP 时，仍然会引用 BOOTP，因此你会在 Packets Detail 面板中看到 BOOTP 协议，而不是 DHCP。但无论如何，我在这本书中仍会将其叫作数据包的 DHCP 部分。

这是一个请求数据包，在消息类型域中标识为 1。发现数据包中的大多数字段或者为空（就像是 IP 地址域）；或者根据上一节所列出的 DHCP 字段，就已经解释的很清楚了。这个数据包的主要内容是这 4 个字段。

DHCP 消息类型：这里的选项类型为 53（t=53），长度为 1，它的值为 1。这些值表明这是一个 DHCP 发现数据包。

客户端标识符：这里提供了客户端请求 IP 地址的额外信息。

所请求 IP 地址：这里提供了客户端希望得到的 IP 地址（通常是之前用过的 IP 地址）。

请求参数列表：这里列出了客户端希望从 DHCP 服务器接收到的不同配置项（其他重要网络设备的 IP 地址）。

2. 提供数据包

这个文件的第二个数据包在 IP 头中列出了可用的 IP 地址，显示这个数据包从 192.168.0.1 发往 192.168.0.10，如图 9-4 所示。因为客户端实际上还没有 192.168.0.10 这个地址，所以服务器会首先尝试使用由 ARP 提供的客户端硬件地址与之通信。如果通信失败，那么它将会直接将提供（Offer）广播出去，进行通信。

第二个数据包的 DHCP 部分，称为提供数据包，表明这是一个响应的消息类型。这个数据包包含了和前一个数据包相同的事务 ID，也就是告诉我们这个响应与我们原先的请求相对应。

该提供数据包由 DHCP 服务器发出，用以向客户端提供其服务。它提供了关于其自身的信息，以及它想要给客户端提供的地址。图 9-4 中，在“你的”（客户端）IP 地址字段中的 IP 地址 192.168.0.10 就是要提供给客户端的。下一个服务器 IP 地址（Next Server IP Address）域中的值 192.168.0.1 表明我们的 DHCP 服务器与默认网关共享一个 IP 地址。

列出的第一个选项指明这个数据包是一台 DHCP 服务器提供的。服务器所提供的接下来的这些选项和客户端的 IP 地址，一起给出了它所能提供的额外信

异步社区Kensuke(18319066421) 专享 请尊重版权

息。你可以看到它给出了如下信息。

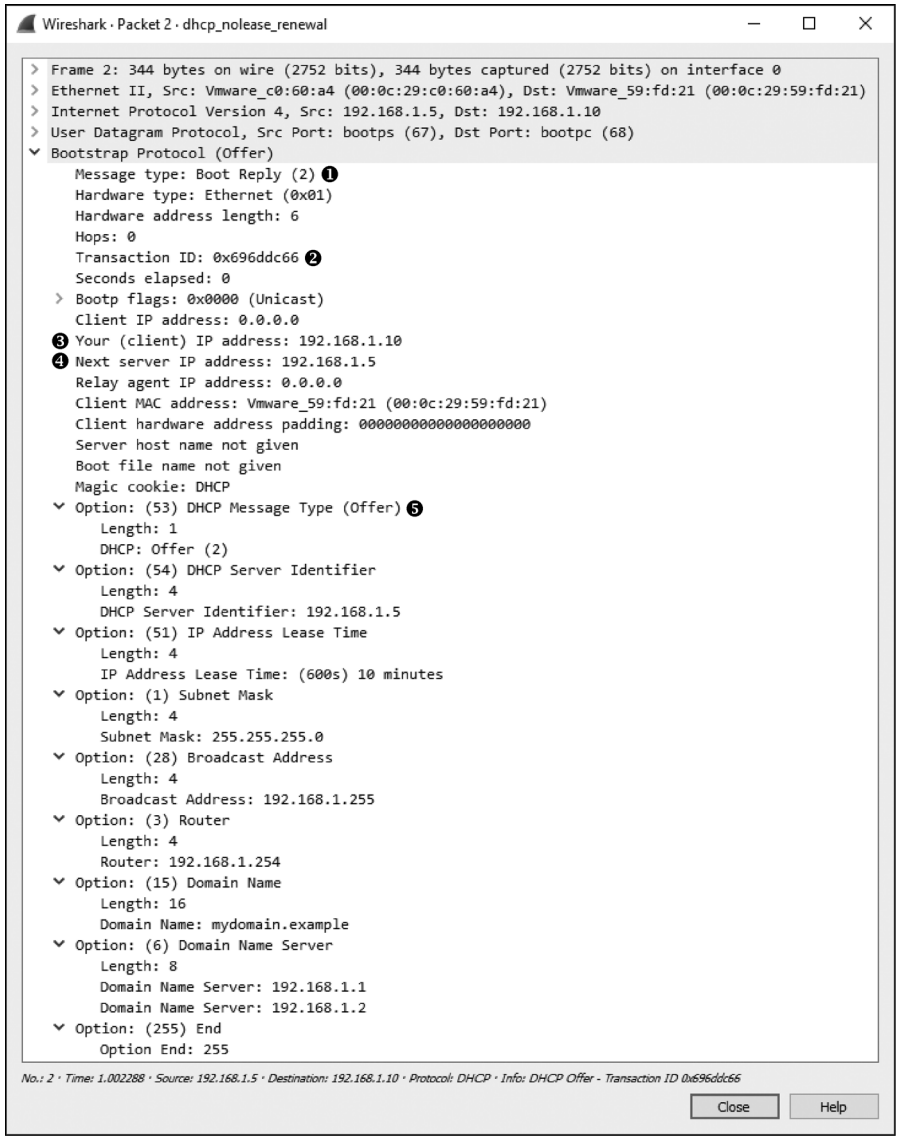


图 9-4　DHCP 提供数据包

- 子网掩码是 255.255.255.0。
- 续租时间是 30min。
- 重新绑定时间的值是 52min30s。

异步社区Kensuke(18319066421) 专享 请尊重版权

- IP 地址的租期是 1h。
- DHCP 服务器的标识符是 192.168.0.1。

3. 请求数据包

在客户端接收到 DHCP 服务器的提供数据包之后，它将以一个 DHCP 请求数据包作为接收确认，如图 9-5 所示。

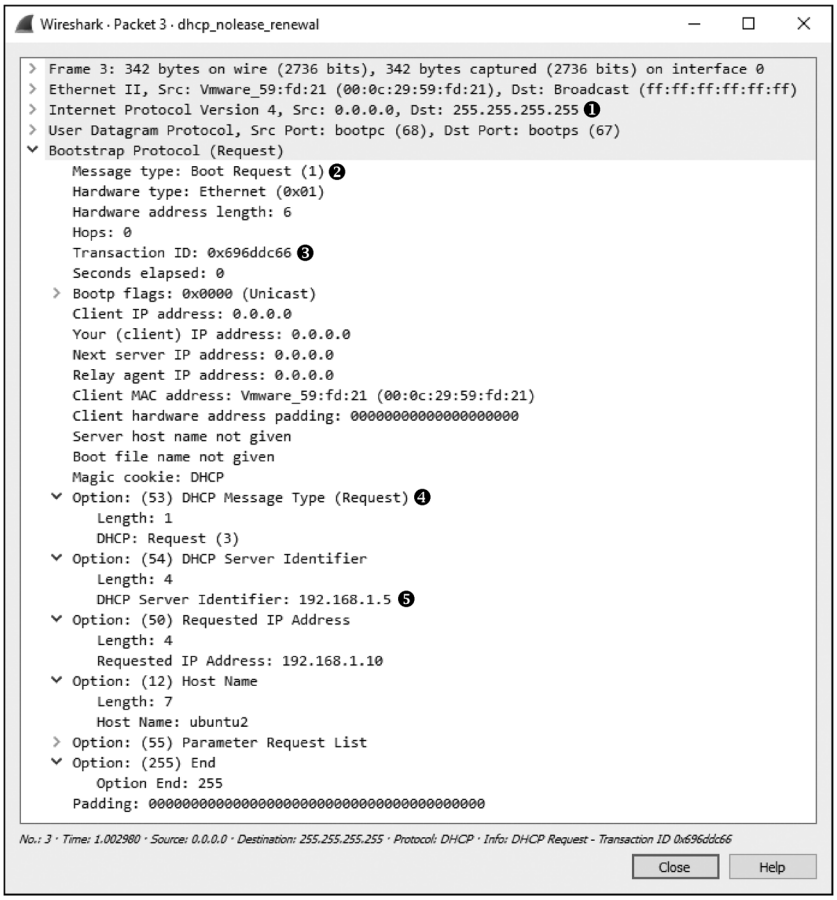


图 9-5　DHCP 请求数据包

这个捕获文件中的第三个数据包仍然从 IP 地址 0.0.0.0 处发出，因为我们还没有完成获取 IP 地址的过程。但数据包现在知道了它所要通信的 DHCP 服务器。

消息类型字段显示这是一个请求数据包。虽然这个捕获文件上的每个数据包都属于同一个续租过程，但因为这是一个新的请求/响应过程，所以它有了一

异步社区Kensuke(18319066421) 专享 请尊重版权

个新的事务 ID。这个数据包与发现数据包相似，其所有的 IP 地址信息都是空的。

在最后的选项域，我们看到这是一个 DHCP 请求。值得注意的是，这个所要请求的 IP 地址不再是空，并且 DHCP 服务器标识符域也填有 IP 地址。

4. 确认数据包

这个过程的最后一步就是 DHCP 在确认数据包中给客户端发送其所请求的 IP 地址，并在其数据库中记录相关信息，如图 9-6 所示。

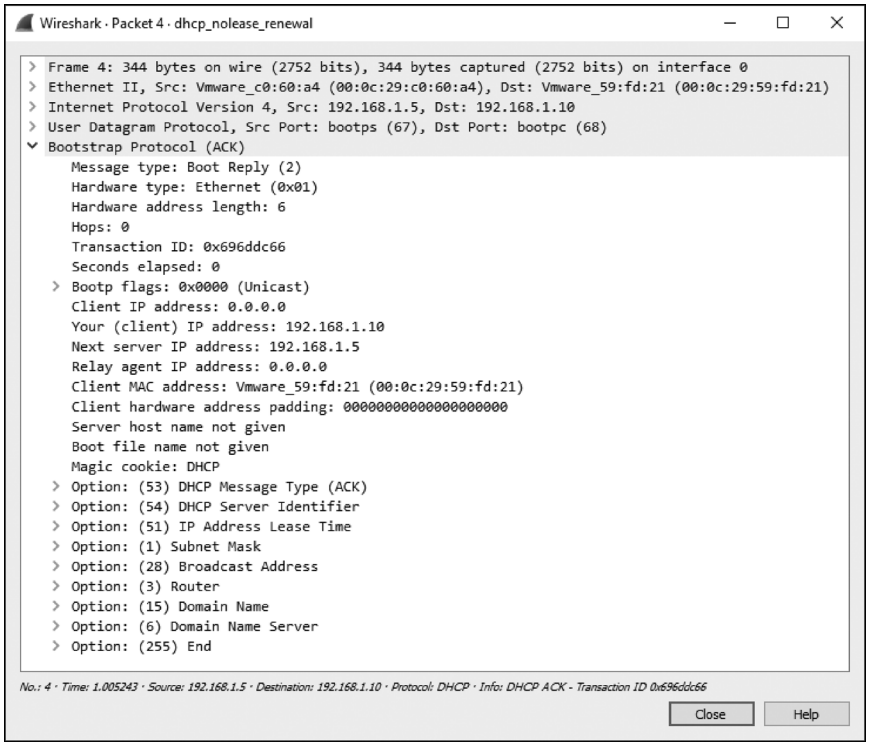


图 9-6　DHCP 确认数据包

这时客户端就有了一个 IP 地址，并且可以用它在网络上通信。

9.1.3　DHCP 租约内续租

当 DHCP 给一台设备分配了一个 IP 地址时，它同时也给客户端定下了一个租约。也就是说客户端在有限的时间之内只用这个 IP 地址，否则就必须续租。我们刚刚介绍的 DORA 过程是出现在客户端第一次获取 IP 地址或

异步社区Kensuke(18319066421) 专享 请尊重版权

者其租约时间已经过期的情况下。在这两种情况下，这台设备都会被视为租约过期。

当一个拥有 IP 地址的客户端在租约内重新启动时，它必须进行一次精简版的 DORA 过程，来重新认领它的 IP 地址。这个过程被称为租约内续租。

在租约内续租时，发现和提供数据包就变得没有必要了。将租约内续租看作是租约过期续租中使用的同种 DORA 过程，可以发现在租约内续租并不需要那么做，而只需完成请求和确认两个步骤就可以了。你可以在文件 dhcp_inlease_renewal.pcap 中看到租约内续租的一个捕获样例。

9.1.4 DHCP 选项和消息类型

DHCP 依赖于其可选项来提供真正的灵活性。正如你所看到的那样，数据包的 DHCP 选项在大小和内容上都可以变化。数据包的整体大小则取决于其所使用的选项。

所有 DHCP 数据包都需要的唯一选项就是消息类型选项（选项 53）。这个选项标识着 DHCP 客户端或者服务器将如何处理数据包中的信息。在表 9-1 中给出了所定义的 8 种消息类型。

表 9-1 DHCP 消息类型

类型号	消息类型	描述
1	发现	客户端用来定位可用的 DHCP 服务器
2	提供	服务器用来给客户端发送发现数据包的响应
3	请求	客户端用来请求服务器所提供的参数
4	拒绝	客户端向服务器指明数据包的无效参数
5	ACK	服务器向客户端发送所请求的配置参数
6	NAK	客户端向服务器拒绝其配置参数的请求
7	释放	客户端向服务器通过取消配置参数来取消租约
8	通知	当客户端已经有 IP 地址时客户端向服务器请求配置参数

9.1.5 DHCP Version6 (DHCPv6)

如果按照表 9-1 对 DHCP 数据包结构的定义，你会发现结构体中没有为 IPv6 地址提供足够的空间。为了解决这一问题，同时又不对 DHCP 协议进行改动，在 RFC3315 中提出了 DHCPv6 协议。由于 DHCPv6 不是基于 BOOTP 协议（DHCP

异步社区Kensuke(18319066421) 专享 请尊重版权

协议的前身）设计的，因此 DHCPv6 的数据包结构要比 DHCP 协议精简很多（见图 9-7）。

动态主机配置协议Version 6 (DHCPv6)					
偏移位	八位组	0	1	2	3
八位组	位	0–7	8–15	16–23	24–31
0	0	消息类型	事务ID		
4+	32+	选项			

图 9-7　DHCPv6 数据包结构

如图 9-7 所示，DHCPv6 数据包结构仅包含 2 个固定字段，其作用与 DHCP 中的类似字段相同。数据包的其他部分取决于位于第一个字节的消息类型。在选项部分，每个选项由一个 2 字节的选项码和一个长度为 2 字节的选项值字段组成。

DHCPv6 实现了和 DHCP 相同的功能，但要理解 DHCPv6 通信，我们必须把 DORA（Discover—Offer—Request—Acknowledgment）替换为 SARR（Solicit—Advertise—Request—Reply）。这一过程如图 9-8 所示，描述了客户端的续租流程。

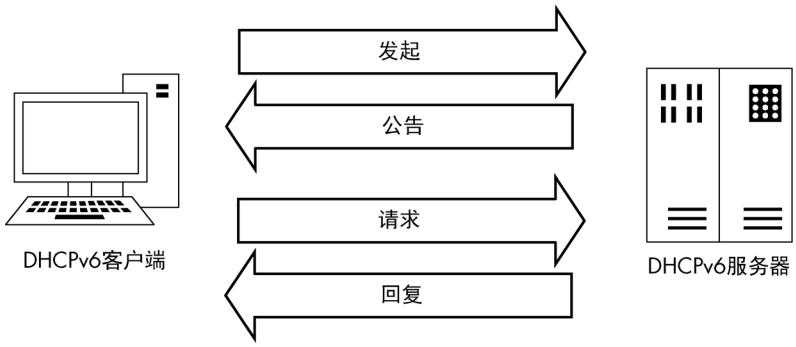


图 9-8　DHCPv6 SARR 续租过程

SARR 过程包括以下 4 个步骤。

（1）发起（Solicit）：客户端向一个特定的组播地址（ff02::1:2）发送一个初始化数据包，尝试在网络上发现可用的 DHCPv6 服务器。

（2）公告（Advertise）：一个可用的 DHCPv6 服务器直接回复客户端，表明此服务器能够提供地址分配和设置服务。

异步社区Kensuke(18319066421) 专享 请尊重版权

（3）请求（Request）：客户端通过组播方式向服务器发起地址配置信息请求。

（4）回复（Reply）：服务器向客户端直接发送其请求的所有配置信息，SARR过程完成。

SARR 过程的概要如图 9-9 所示，这部分数据包从 dhcp6_outlease_acquisition.pcapng 中提取。在这个例子中，一台新接入网络的主机（fe80::20c:29ff:fe5e:7744）按照 SARR 过程，从 DHCPv6 服务器（fe80::20c:29ff:fe1f:a755）获取配置信息。每一个数据包代表了 SARR 过程中的一个步骤，其中，初始的 solicit（发起）和 advertise（公告）数据包的事务 ID 为 0x9de03f，request（请求）和 reply（回复）数据包的事务 ID 为 0x2d1603。这个通信过程使用 546 和 547 端口，这两个端口是 DHCPv6 使用的标准端口。

No	Time	Source	Destination	Protocol	Length	Info
1	0.…	fe80::20c:29ff:fe5e:7744	ff02::1:2	DHCPv6	118	Solicit XID: 0x9de03f CID: 000100011def69bd000c295e7744
2	0.…	fe80::20c:29ff:fe1f:a755	fe80::20c:29ff:fe5e:7744	DHCPv6	166	Advertise XID: 0x9de03f CID: 000100011def69bd000c295e7744 IAA: 2001:db8:1:2::1002
3	1.…	fe80::20c:29ff:fe5e:7744	ff02::1:2	DHCPv6	164	Request XID: 0x2d1603 CID: 000100011def69bd000c295e7744 IAA: 2001:db8:1:2::1002
4	1.…	fe80::20c:29ff:fe1f:a755	fe80::20c:29ff:fe5e:7744	DHCPv6	166	Reply XID: 0x2d1603 CID: 000100011def69bd000c295e7744 IAA: 2001:db8:1:2::1002

图 9-9　客户端通过 DHCPv6 获得一个 IPv6 地址

总的来说，DHCPv6 和 DHCP 的数据包结构有很大区别，但是在功能实现思路上是一致的。这个过程仍然包括 DHCP 服务器发现步骤和正式的配置信息获取步骤。这些事件通过客户端和服务器之间交互数据包中的事务 ID 进行关联。传统的 DHCP 机制不支持 IPv6 地址分配，因此，如果你的设备能够从网络中的某个服务器自动获取 IPv6 地址，则这表明你的网络中已经在运行 DHCPv6 服务。如果你想要进一步比较 DHCP 和 DHCPv6，我们建议你使用抓包工具在客户端和服务器端逐步进行分析。

9.2　域名系统

域名系统（Domain Name System, DNS）是最重要的互联网协议之一，因为它是众所周知的黏合剂。DNS 将例如 Google 的域名和 74.125.159.99 的 IP 地址捆绑起来。当我们想要和一台网络设备通信却不知道它的 IP 地址时，就可以使用它的 DNS 名字来进行访问。

DNS 服务器存储了一个有着 IP 地址和 DNS 名字映射资源记录的数据库，并将其和客户端以及其他 DNS 服务器共享。

注意　由于 DNS 服务器的结构很复杂，因此我们只关注于通常类型的 DNS 流量。

异步社区Kensuke(18319066421) 专享 请尊重版权

9.2.1 DNS 数据包结构

如图 9-10 所示，DNS 数据包和我们之前所看到的数据包类型结构有所不同。DNS 数据包中会出现下面的一些域。

<table>
<tr><td colspan="11">域名系统 (DNS)</td></tr>
<tr><td>偏移位</td><td>八位组</td><td colspan="3">0</td><td colspan="3">1</td><td colspan="2">2</td><td colspan="2">3</td></tr>
<tr><td>八位组</td><td>位</td><td colspan="3">0–7</td><td colspan="3">8–15</td><td colspan="2">16–23</td><td colspan="2">24–31</td></tr>
<tr><td>0</td><td>0</td><td colspan="3">DNS ID号</td><td>QR</td><td>操作码</td><td>AA</td><td>TC</td><td>RD</td><td>RA</td><td>Z</td><td>响应码</td></tr>
<tr><td>4</td><td>32</td><td colspan="3">问题计数</td><td colspan="4">回答计数</td><td colspan="2"></td></tr>
<tr><td>8</td><td>64</td><td colspan="3">域名服务器计数</td><td colspan="4">额外记录计数</td><td colspan="2"></td></tr>
<tr><td>12+</td><td>96+</td><td colspan="3">问题区段</td><td colspan="4">回答区段</td><td colspan="2"></td></tr>
<tr><td></td><td></td><td colspan="3">权威区段</td><td colspan="4">额外信息区段</td><td colspan="2"></td></tr>
</table>

图 9-10　DNS 数据包结构

DNS ID 号（DNS ID Number）：用来对应 DNS 查询和 DNS 响应。

查询/响应（Query/Response, QR）：用来指明这个数据包是 DNS 查询还是响应。

操作码（OpCode）：用来定义消息中请求的类型。

权威应答（Authoritative Answer, AA）：如果响应数据包中设定了这个值，则说明这个响应是由域内权威域名服务器发出的。

截断（Truncation, TC）：用来指明这个响应由于太长，无法装入数据包而被截断。

期望递归（Recursion Desired, RD）：如果在请求中设定了这个值，则说明 DNS 客户端在目标域名服务器不含有所请求信息的情况下，要求进行递归查询。

可用递归（Recursion Available, RA）：如果响应中设定了这个值，则说明域名服务器支持递归查询。

保留（Z）：在 RFC1035 的规定中被设为全 0，但有时会被用来作为 RCode 域的扩展。

响应码（Response Code）：在 DNS 响应中用来指明错误。

异步社区Kensuke(18319066421) 专享 请尊重版权

问题计数（Question Count）：在问题区段中的条目数。

回答计数（Answer Count）：在回答区段中的条目数。

域名服务器计数（Name Server Count）：在权威区段的域名资源记录数。

额外记录计数（Additional Records Count）：在额外信息区段中的其他资源记录数。

问题区段（Question section）：大小可变、包含要被发送到 DNS 服务器的一条或多条的信息查询的部分。

回答区段（Answer section）：大小可变、包含用来回答查询的一条或多条资源记录。

权威区段（Authority section）：大小可变、包含指向权威域名服务器的资源记录，用以继续解析过程。

额外信息区段（Additional Information section）：大小可变、包含与查询有关的额外信息，但对于回答查询这并不是绝对必要的资源记录。

9.2.2 一次简单的 DNS 查询过程

DNS 以查询/响应的模式工作。当一个客户端想要将一个 DNS 名字解析成 IP 地址时，它会向 DNS 服务器发送一个查询，然后服务器在响应中提供所请求的信息。在较简单的情形下，这个过程包含着两个数据包，正如在捕获文件 dns_query_response.pcap 中所看到的那样。

第一个数据包如图 9-11 所示，是由 192.168.0.114 的客户端通过 DNS 的标准 25 端口发向 205.152.37.23 的服务器的 DNS 查询。

当检查这个数据包的头部时，你会发现 DNS 也是基于 UDP 协议的。

在数据包的 DNS 区段，你可以看到数据包开头的一些较小域都被 Wireshark 合并成为了一个标志区段（Flags section）。展开这个区段，你会看到这个消息是一个典型的请求：没有被截断并且期望递归查询（我们随后将介绍递归查询）。在展开查询区段时，里面也仅有一个问题。这个问题是查询名字为 wireshark.org 的主机类型（type A）互联网（IN）地址。这个数据包基本就是在问：“哪个 IP 地址对应着 wireshark.org 域？”

数据包 2 响应了这个请求，如图 9-12 所示。因为这个数据包拥有唯一的标识码，所以我们知道这里包含着对于原始查询的正确响应。

异步社区Kensuke(18319066421) 专享 请尊重版权

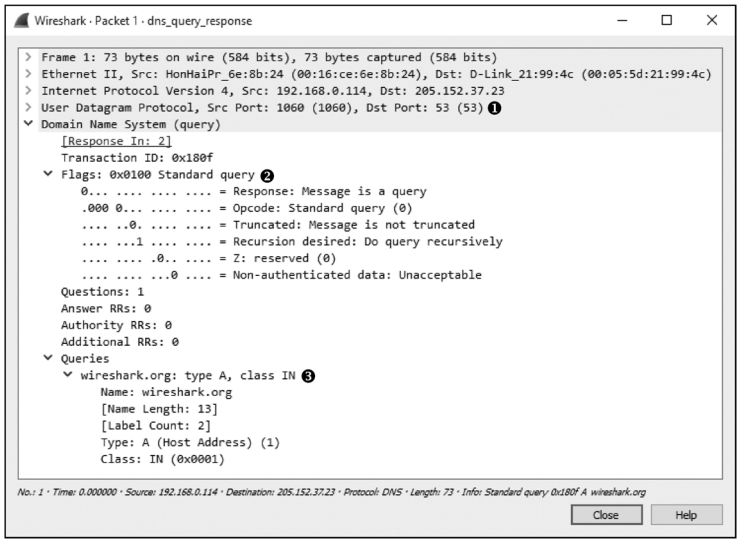


图 9-11　DNS 查询数据包

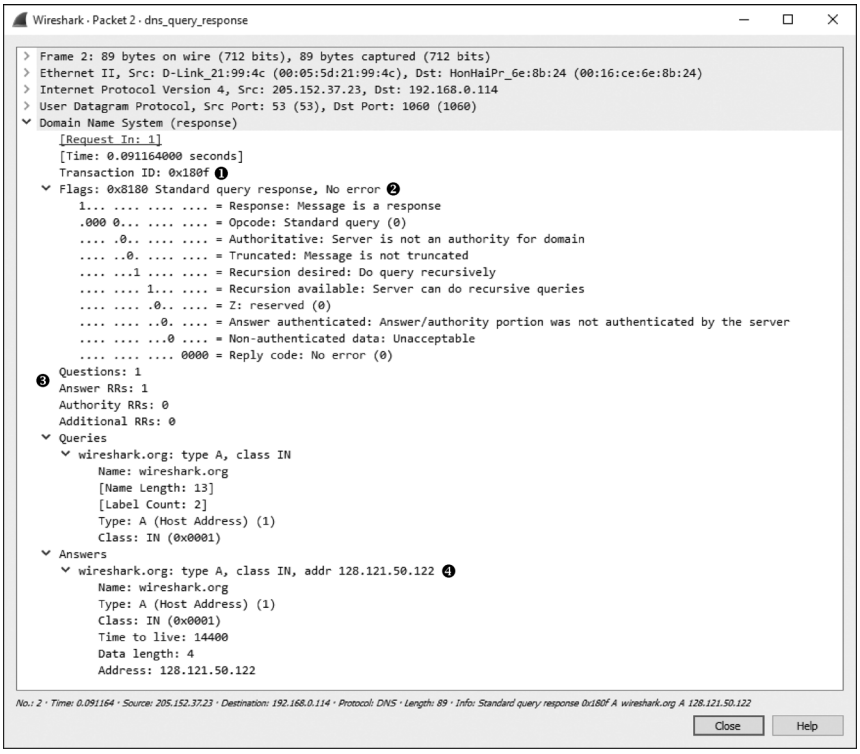


图 9-12　DNS 响应数据包

异步社区Kensuke(18319066421) 专享 请尊重版权

标志区段可以确保这是一个响应并且允许必要的递归。这个数据包仅包含一个问题和一个资源记录，因为它将原问题和回答连接了起来。展开回答区段可以看到对于查询的回答：wireshark.org 的地址是 128.121.50.122。有了这个信息，客户端就可以开始构建 IP 数据包，并与 wireshark.org 进行通信了。

9.2.3 DNS 问题类型

DNS 查询和响应中所使用的类型域，指明了这个查询或者响应的资源记录类型。表 9-2 中列出了一些常用的消息/资源记录类型。在正常流量和本书中，你将会看到这些类型。

表 9-2 常用 DNS 资源记录类型

值	类型	描述
1	A	IPv4 主机地址
2	NS	权威域名服务器
5	CNAME	规范别名
15	MX	邮件交换
16	TXT	文本字符串
28	AAAA	IPv6 主机地址
251	IXFR	增量区域传送
252	AXFR	完整区域传送

9.2.4 DNS 递归

由于互联网的 DNS 结构是层级式的，因此为了能够回答客户端提交的查询，DNS 服务器必须能够彼此通信。我们的内部 DNS 服务器知道本地局域网服务器的名字和 IP 地址的映射，但不太可能知道 Google 或者 Dell 的 IP 地址。

当 DNS 服务器需要查找一个 IP 地址时，它会代表发出请求的客户端向另一个 DNS 服务器进行查询。实际上，这个 DNS 服务器与客户端的行为相同。这个过程叫作递归查询。

打开文件 dns_recursivequery_client.pcap，可以分别看到 DNS 客户端和服务

异步社区Kensuke(18319066421) 专享 请尊重版权

器视角的递归查询过程。这个文件包含了捕获客户端 DNS 流量文件的两个数据包。第一个数据包是从 DNS 客户端 172.16.0.8 发往 DNS 服务器 172.16.0.102 的初始查询，如图 9-13 所示。

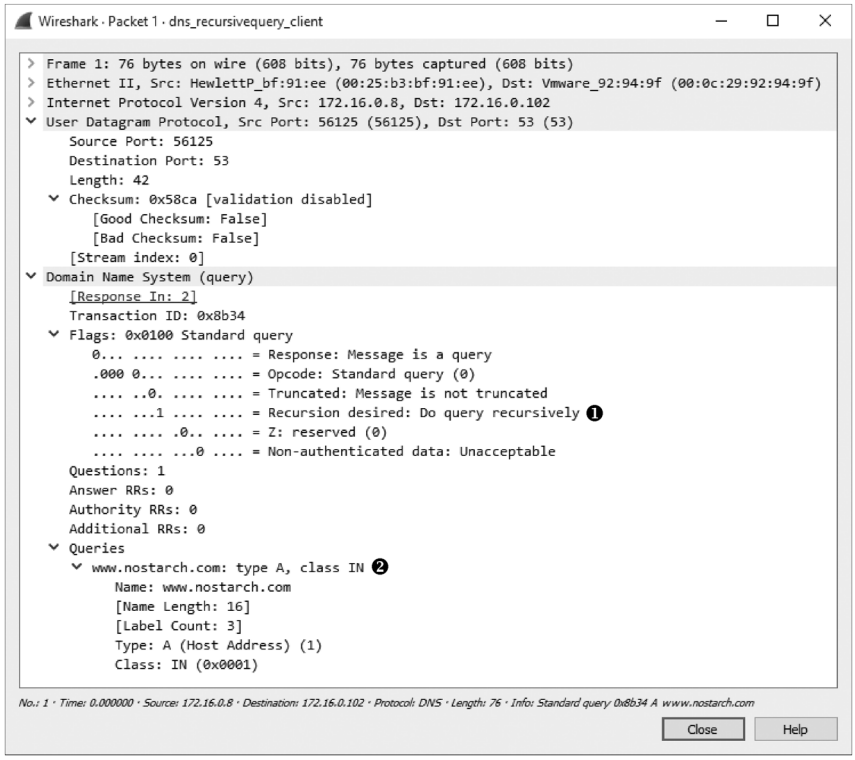


图 9-13　设置有期望递归位的 DNS 查询

当你展开这个数据包的 DNS 区段时，可以看到这是一个用以查找 DNS 名称 www.nostarch.com 的 A 类型记录的标准查询。展开标志区段，可以了解更多关于这个数据包的信息，你可以看到期望递归的标志。

第二个数据包是我们所希望看到的对于初始数据包的响应，如图 9-14 所示。

这个数据包的事务 ID 和我们的查询相匹配，也没有列出错误，所以我们就得到了 www.nostarch.com 所对应的 A 类型资源记录。

异步社区Kensuke(18319066421) 专享 请尊重版权

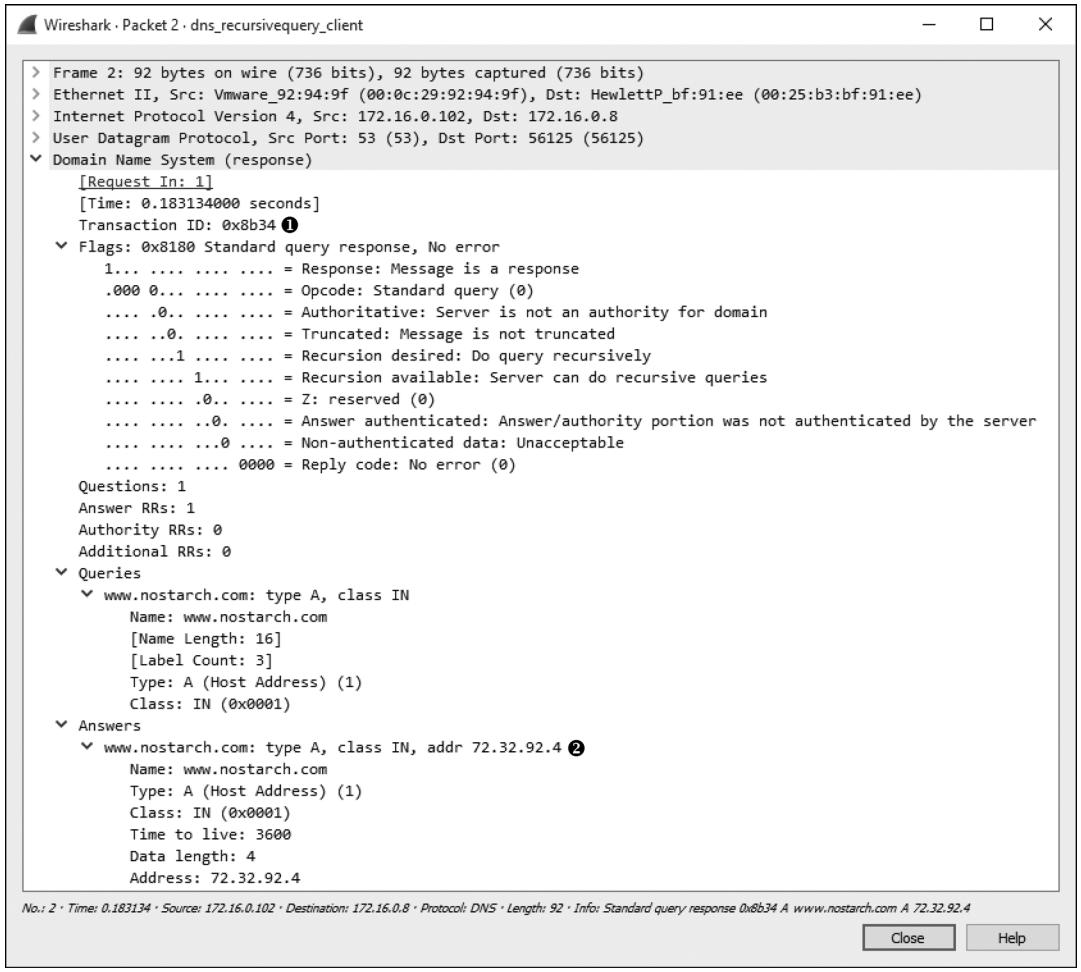
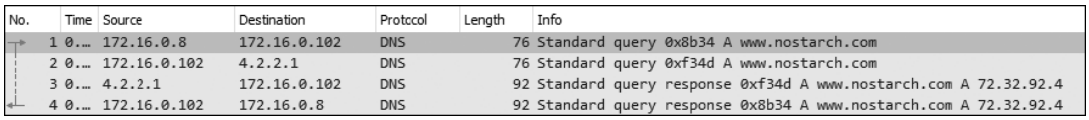


图 9-14　DNS 查询响应

如果我们想要知道查询是否被递归应答，唯一的方法就是当进行递归查询时监听 DNS 服务器的流量，正如文件 dns_recursivequery_server.pcap 中所示的那样。这个文件显示了查询进行时本地 DNS 服务器流量的捕获，如图 9-15 所示。第一个数据包和我们前一个捕获文件中的初始查询相同。这时，DNS 服务器接到了这个查询，在其本地数据包检查后，发现它并不知道关于 DNS 域名（nostarch.com）所对应 IP 地址这个问题的答案。由于这个数据包发送时设置了期望递归，因此你会在第二个数据包中看到，这个 DNS 服务器为了得到答案向其他 DNS 服务器询问这个问题。

异步社区Kensuke(18319066421) 专享 请尊重版权



No.	Time	Source	Destination	Protocol	Length	Info
1	0.…	172.16.0.8	172.16.0.102	DNS	76	Standard query 0x8b34 A www.nostarch.com
2	0.…	172.16.0.102	4.2.2.1	DNS	76	Standard query 0xf34d A www.nostarch.com
3	0.…	4.2.2.1	172.16.0.102	DNS	92	Standard query response 0xf34d A www.nostarch.com A 72.32.92.4
4	0.…	172.16.0.102	172.16.0.8	DNS	92	Standard query response 0x8b34 A www.nostarch.com A 72.32.92.4

图 9-15　从服务器的角度进行 DNS 查询

在第二个数据包中，位于 172.16.0.102 的 DNS 服务器向 4.2.2.1，也就是其所设定的要转发上行请求的服务器，发送了一个新的查询，如图 9-16 所示。这个请求是原始的镜像，并将 DNS 服务器变成一个客户端。

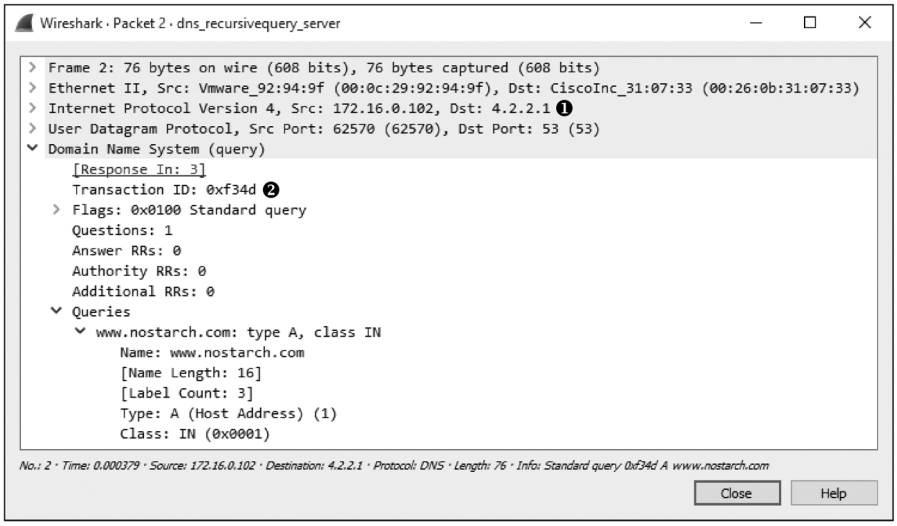


图 9-16　递归 DNS 查询

由于这个事务 ID 与之前捕获文件中的事务 ID 不同，因此我们可以将它作为一个新的查询。在这个数据包被服务器 4.2.2.1 接收之后，本地 DNS 服务器就接到了响应，如图 6-17 所示。

接到了这个响应后，本地 DNS 服务器就可以将第四个，也就是最后一个带有请求信息的数据包传递给 DNS 客户端。

虽然这个例子只展示了一层的递归，但对一个 DNS 请求来说递归查询可能会发生很多次。这里我们接到了来自 DNS 服务器 4.2.2.1 的回答，但那个服务器可能为了寻找答案也向其他服务器进行了递归查询。一个简单查询在其得到最终响应之前可能遍历了全世界。图 9-18 展示了递归 DNS 查询的过程。

异步社区Kensuke(18319066421) 专享 请尊重版权

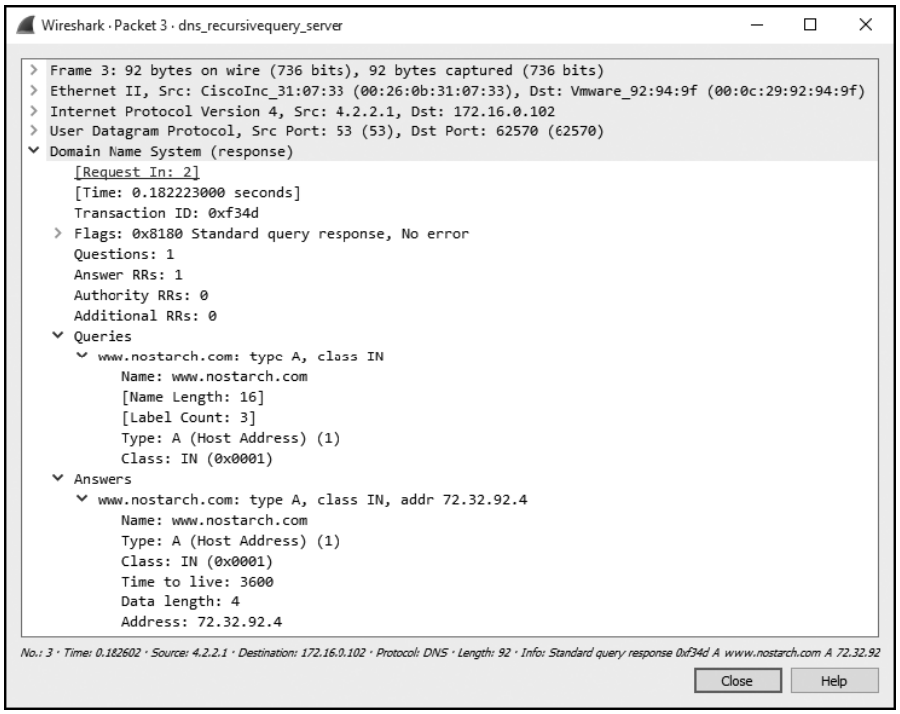


图 9-17　对递归 DNS 查询的响应

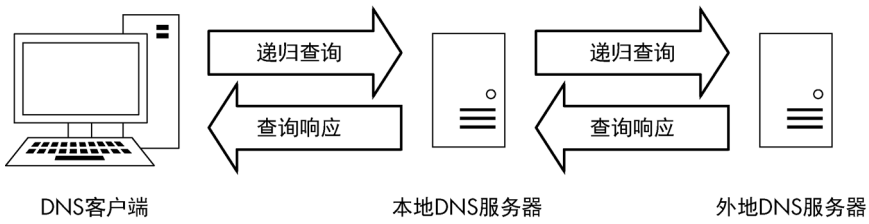


图 9-18　DNS 递归查询

9.2.5　DNS 区域传送

DNS 区域是一个 DNS 服务器所授权管理的名字空间（或是一组 DNS 名称）。举例来说，Emma’s Diner 这个网站可能由一个 DNS 服务器对 emmasdiner.com 负责。这样，无论是 Emma’s Diner 内部或者外部的设备，如果希望将 emmasdiner.com 解析成 IP 地址，都需要和这个区域的权威，也就是这个 DNS 服务器联系。如果 Emma’s Diner 发展壮大了，它可能会增加一个 DNS 服务器，专门用来处理其名字

异步社区Kensuke(18319066421) 专享 请尊重版权

空间的 email 部分，比如 mail.emmasdiner.com，那么这个服务器，就成为这个邮件子区域的权威。如果必要的话，还可以为子域名添加更多的 DNS 服务器，如图 9-19 所示。

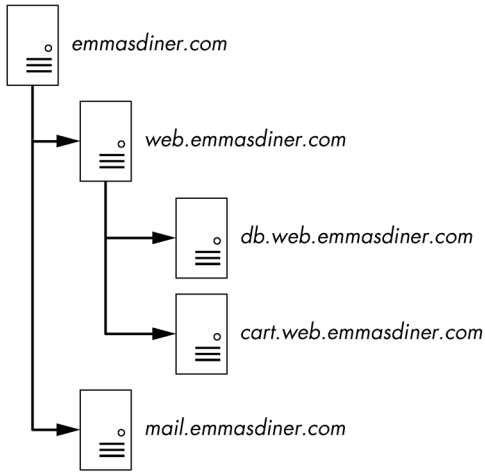


图 9-19　DNS 区域划分名称空间的责任

区域传送指的是通常出于冗余备份的需要，在两台设备之间传送区域数据。举例来说，在拥有多个 DNS 服务器的组织中，管理员通常都会配置一台备用 DNS 服务器，用来维护一份主服务器 DNS 信息的副本，以防止主 DNS 服务器不可用。主要存在两种区域传送。

完整区域传送（AXFR）：这个类型的传送将整个区域在设备间进行传送。

增量区域传送（IXFR）：这个类型的传送仅传送区域信息的一部分。

文件 dns_axfr.pcap 包含了一个主机 172.16.16.164 和 172.16.16.139 之间进行完整区域传送的例子。

当第一眼看这个文件时，你可能会怀疑是否开错了文件，因为你所见到的是 TCP 数据包而不是 UDP 数据包。虽然 DNS 基于 UDP 协议，但它在比如区域传送的一些任务中也会使用 TCP 协议，因为 TCP 对于规模数据的传输更加可靠。这个捕获文件中的前 3 个数据包是 TCP 的三次握手。

第四个数据包开始在 172.16.16.164 和 172.16.16.139 之间进行实际的区域传送。这个数据包并不包含任何 DNS 信息。由于区域传送请求的数据包中的数据由多个数据包所发送，因此这个数据包被标记为重组装 PDU 的 TCP 分片。数据包 4 和 6 包含了数据包的数据。数据包 5 是对于数据包 4 被成功接收的确认。这些数据包以这种方式显示出来是因为 Wireshark 出于可读性的考虑将 TCP 数据包如此解析并呈现。这里我们可以将数据包 6 作为完整的 DNS 区域传送请求，如图 9-20 所示。

区域传送请求是典型的查询，但它请求的是 AXFR 类型而不是单一记录类型，这意味着它希望从服务器接收全部 DNS 区域。服务器在数据包 7 中回复了区域记录，如图 9-21 所示。正如你所见到的那样，区域传送包含了相当多的数

异步社区Kensuke(18319066421) 专享 请尊重版权

据，并且这还是一个很简单的例子！在区域传送完成之后，捕获文件以 TCP 连接的终止过程作为结束。

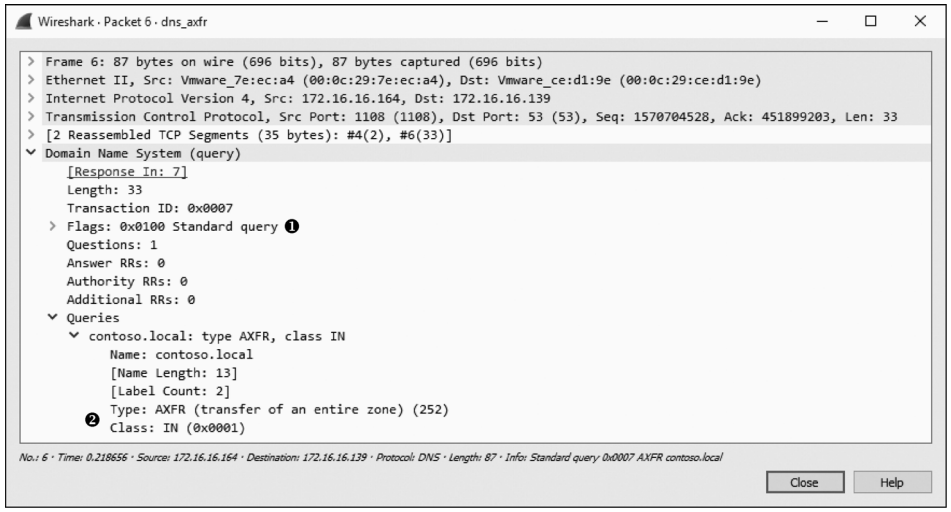


图 9-20　DNS 完整区域传送请求

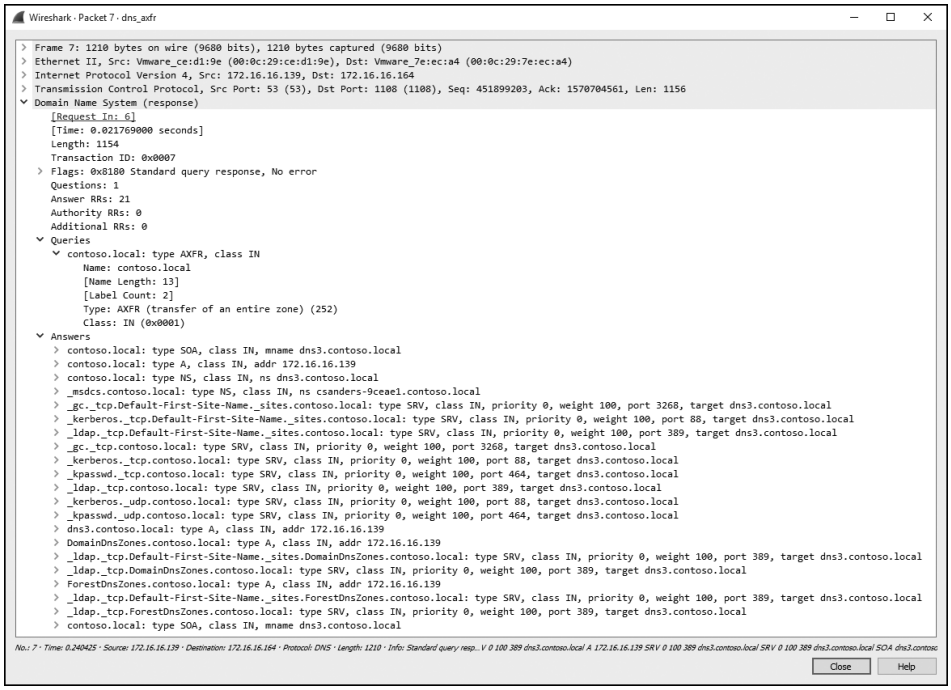


图 9-21　正在进行的 DNS 完整区域传送

异步社区Kensuke(18319066421) 专享 请尊重版权

警告　区域传送的数据如果落入他人手中可能会很危险。举例来说，通过枚举一个 DNS 服务器，你可以绘出整个网络的基础结构。

9.3 超文本传输协议

超文本传输协议（Hypertext Transfer Protocol, HTTP）是万维网（World Wide Web）的传输机制，允许浏览器通过连接 Web 服务器浏览网页。目前在大多数组织中，HTTP 流量在网络中所占的比率是最高的。每一次使用 Google 搜索、连接 Twitter、发一条微博，或者在 ESPN 上查看肯塔基大学的篮球比分，你都会用到 HTTP。

我们不会去看 HTTP 传输的数据包结构，因为有着不同目的的数据包的内容差别会很大。这个任务就留给你了。这里，我们来看 HTTP 的实际应用。

9.3.1 使用 HTTP 浏览

HTTP 常被用来浏览 Web 服务器上使用浏览器访问的网页。捕获文件 http_google.pcap 就给出了这样一个使用 TCP 作为传输层协议的 HTTP 传输的例子。通信以客户端 172.16.16.128 和 Google 的 Web 服务器 74.125.95.104 的三次握手开始。

在建立了连接之后，第一个被标为 HTTP 的数据包从客户端发往服务器，如图 9-22 所示。

HTTP 数据包通过 TCP 被传输到服务器的 80 端口，也就是 HTTP 通信的标准端口（8080 端口也常被使用）。

HTTP 数据包会被确定为 8 种不同请求方法中的一种（根据 HTTP 规范版本 1.1 的定义）。这些请求方法指明了数据包发送者想要对接收者采取的动作。如图 9-22 所示，这个数据包的方法是 GET，它请求/作为通用资源标识符（Uniform Resource Indicator），并且请求版本是 HTTP/1.1。这些信息告诉我们这个客户端请求使用 HTTP 的 1.1 版本，下载 Web 服务器的根目录（/）。

接下来，主机向 Web 服务器发送关于自己的信息。这些信息包含了正在使用的用户代理（浏览器）、浏览器接受的语言（Accept Languages）和 Cookie 信

异步社区Kensuke(18319066421) 专享 请尊重版权

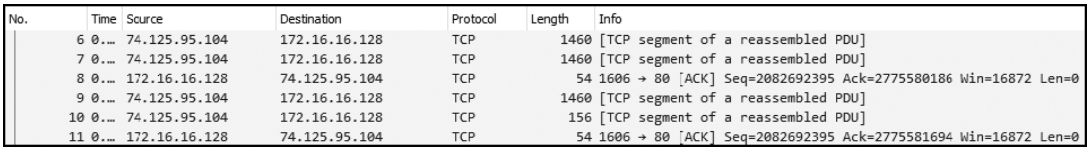
息（位于捕获的底部）。为保证兼容性，服务器可以利用这些信息，决定返回给客户端的数据。



图 9-22　初始 HTTP GET 请求数据包

当服务器接收到了数据包 4 中的 HTTP 请求时，它会响应一个 TCP ACK，用以确认数据包，并在数据包 6～11 中传输所请求的数据。HTTP 只被用来解决客户端和服务器的应用层命令。当进行数据传输时，除了数据流的开始和结束就看不到应用层的控制信息了。

服务器在数据包 6 和 7 中发送数据，数据包 8 是来自客户端的确认，数据包 9 和 10 是另外两个数据包，数据包 11 是另外一个确认，如图 9-23 所示。虽然 HTTP 仍然负责这些传输，但所有这些数据包在 Wireshark 中都被显示为 TCP 分片而不是 HTTP 数据包。



No.	Time	Source	Destination	Protocol	Length	Info
6	0.…	74.125.95.104	172.16.16.128	TCP	1460	[TCP segment of a reassembled PDU]
7	0.…	74.125.95.104	172.16.16.128	TCP	1460	[TCP segment of a reassembled PDU]
8	0.…	172.16.16.128	74.125.95.104	TCP	54	1606 → 80 [ACK] Seq=2082692395 Ack=2775580186 Win=16872 Len=0
9	0.…	74.125.95.104	172.16.16.128	TCP	1460	[TCP segment of a reassembled PDU]
10	0.…	74.125.95.104	172.16.16.128	TCP	156	[TCP segment of a reassembled PDU]
11	0.…	172.16.16.128	74.125.95.104	TCP	54	1606 → 80 [ACK] Seq=2082692395 Ack=2775581694 Win=16872 Len=0

图 9-23　客户端浏览器和 Web 服务器之间在使用 TCP 传输数据

在数据传输结束后，数据的重组装流就已经被发送完了，如图 9-24 所示。

异步社区Kensuke(18319066421) 专享 请尊重版权

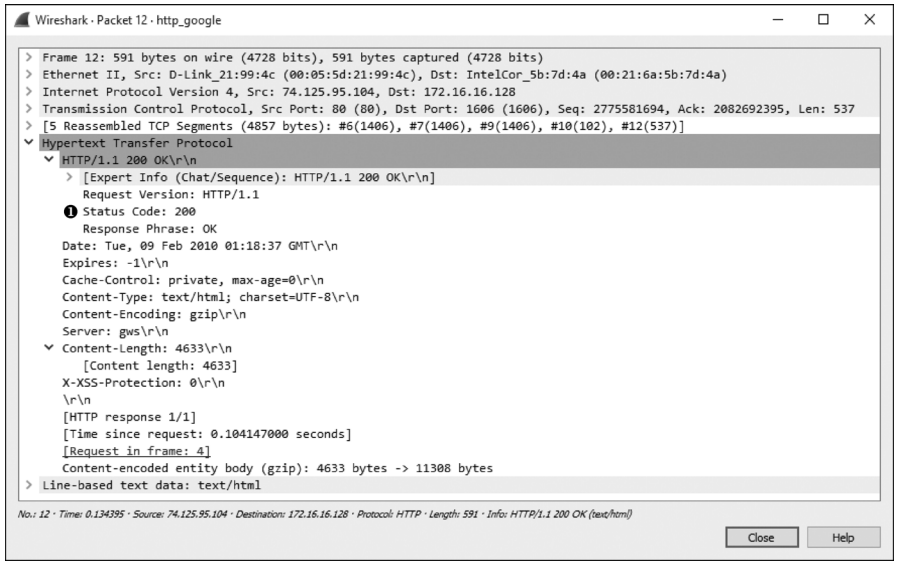


图 9-24　最后有着相应码 200 的 HTTP 数据包

注意　在许多情况下，当浏览包列表时，你无法看到可读的 HTML 数据，因为这些数据被 gzip 压缩以提高带宽效率，这是由 Web 服务器的 HTTP 响应中的内容偏码字段表示的。只有查看完整的流时，数据才能被解码并易于读取。

HTTP 使用了一些预定义的相应码，来表示请求方法的结果。在这个例子中，我们看到一个带有 200 响应码的数据包，表示一次成功的请求方法。这个数据包同样包含一个时间戳，以及一些关于 Web 服务器内容编码和配置参数的额外信息。当客户端接收到这个数据包后，这次处理便完成了。

9.3.2　使用 HTTP 传送数据

我们看过了从 Web 服务器下载数据的过程，现在将注意力转向上传数据。文件 http_post.pcap 包含了一个非常简单的上传的例子：一个用户在一个网站发表评论。在初始的三次握手之后，客户端（172.16.16.128）向 Web 服务器（69.163.176.56）发送了一个 HTTP 数据包，如图 9-25 所示。

这个数据包使用 POST 方法，来向 Web 服务器上传数据以供处理。这里使用的 POST 方法指明了 URI /wp-comments-post.php 以及 HTTP 1.1 请求版本。如果想看上传数据的内容，可以展开这个数据包的 Line-based Text Data 部分。

在这个 POST 的数据传输完成之后，服务器会发送一个 ACK 数据包。如图 9-26

异步社区Kensuke(18319066421) 专享 请尊重版权

所示，服务器在数据包 6 中传输了一个响应码 302（代表“找到”）作为响应。

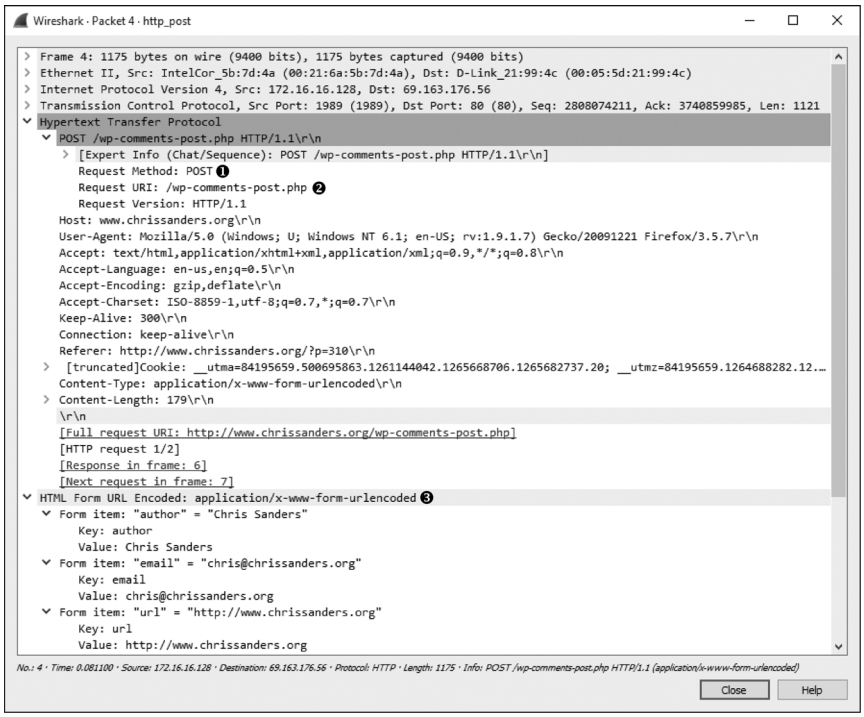


图 9-25　HTTP POST 数据包

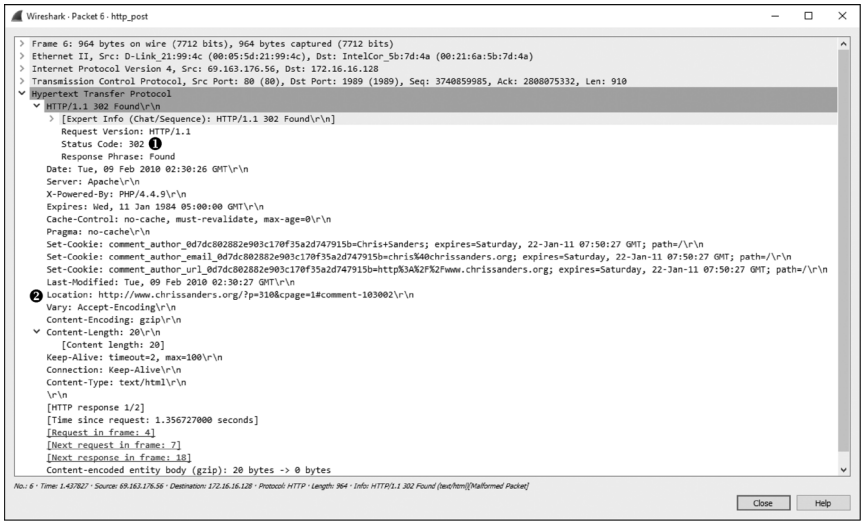


图 9-26　HTTP 响应码 302 用来进行重定向

异步社区Kensuke(18319066421) 专享 请尊重版权

302 相应码是 HTTP 世界的一个常用的重定向手段。这个数据包的 Location 域指明了客户端被重定向的位置。在这里，这个地方便是评论所发表的原始 Web 网页。最后，服务器传送一个状态码 200，并且这个页面的内容会在接下来的一些数据包中进行发送，从而完成传输。

9.4 简单邮件传输协议（SMTP）

如果说 Web 浏览是用户参与次数最多的网络活动，那么收发邮件有可能是第二位。简单邮件传输协议（SMTP）是发送邮件的标准，它被 Microsoft Exchange 和 Postfix 等平台使用。

与 HTTP 一样，SMTP 由于实现方式、与客户端/服务器兼容性相关的一系列特性的不同，在数据包结构上存在多样性。在本节中，我们将通过在数据包层面，对邮件发送过程进行分析，来探究 SMTP 的一些基本功能。

9.4.1 收发邮件

邮件服务架构与邮政服务类似。发信方写了一封邮件，然后将邮件投入发信方的邮箱；邮递员将邮件取走，并将邮件送到邮局进行分拣。从这个邮局出发，这封邮件将被投递至由此邮局提供服务的另一个邮箱，或者转发至另一个负责投递这封邮件的邮局。一封邮件在投递过程中，可能会通过多个邮局，甚至是通过专门用于特定地域内邮局间投递的“Hub”邮局。这个信息传递流程如图 9-27 所示。

电子邮件投递与以上过程非常类似，但是相关术语有少许不同。对于个人用户，电子邮箱取代了物理邮箱，为此用户提供邮件的存储、发送和接收服务。用户通过邮件用户代理（MUA），如 Microsoft Outlook 或 Mozilla Thunderbird，访问邮箱。

当用户发送邮件时，邮件将通过 MUA 发送至邮件传输代理（MTA）。MTA 通常指邮件服务器，流行的邮件服务器应用包括 Microsoft Exchange 和 Postfix。如果邮件的收件方与发件方域名相同，则 MTA 将直接发送邮件至收件人邮箱。如果邮箱被发送至其他域名下的邮箱，则 MTA 将通过 DNS 找到收件方的邮件服务器地址，并将邮件传输至该服务器。值得注意的是，邮件服务器通常还包含其他组件，如邮件发送代理（MDA）和邮件提交代理（MSA），但是从网络的角度看，我们通常只关注客户端和服务器的概念。以上过程的基本概述如图 9-28 所示。

异步社区Kensuke(18319066421) 专享 请尊重版权

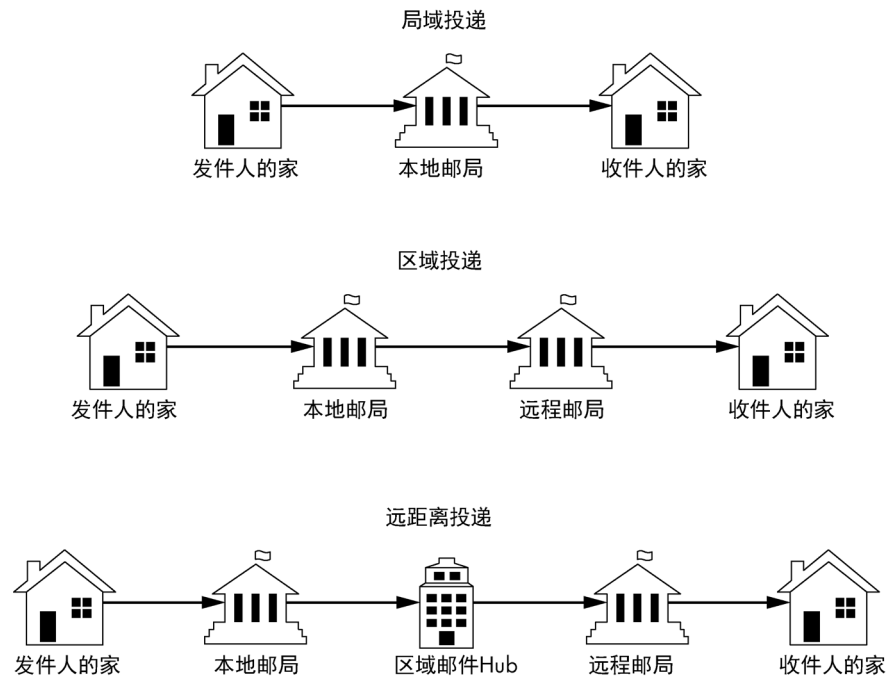


图 9-27　通过邮政服务发送邮件

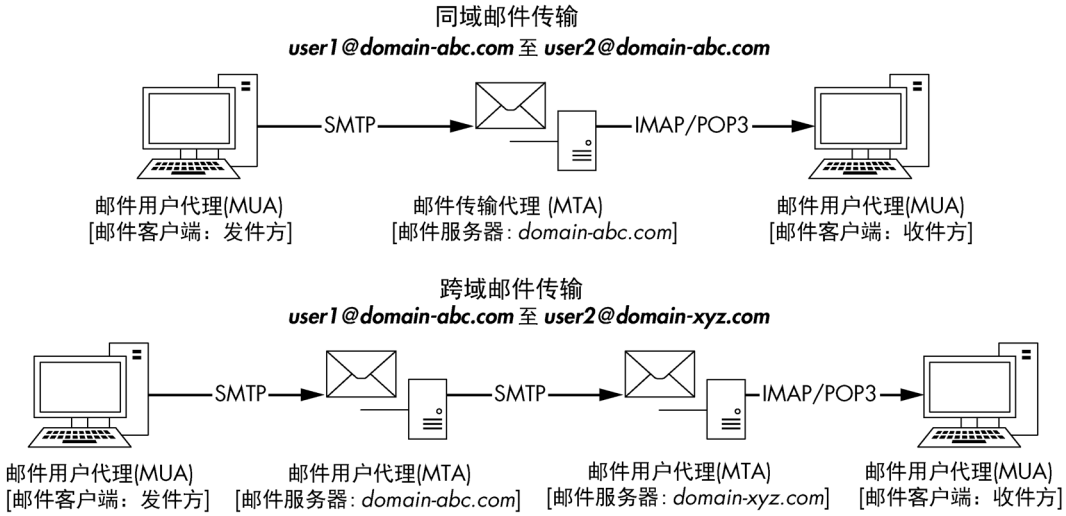


图 9-28　通过 SMTP 发送邮件

出于简化流程的目的，我们用邮件客户端表示 MUA，邮件服务器表示 MTA。

异步社区Kensuke(18319066421) 专享 请尊重版权

9.4.2 跟踪一封电子邮件

对电子邮件如何传输有了基本理解之后，我们开始查看这一过程中的数据包。首先从图 9-29 所描述的场景开始。

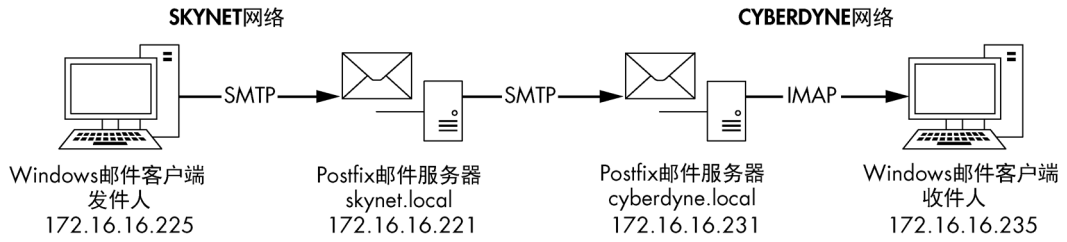


图 9-29　跟踪一封电子邮件从发送方至收件方的传输过程

以上场景分为 3 个步骤。

（1）用户从计算机（172.16.16.225）发送一封电子邮件，邮件客户端使用 SMTP 协议，将邮件传输至本地邮件服务器（172.16.16.221/skynet.local 域）。

（2）本地邮件服务器接收邮件，并使用 SMTP 将其传输至远程邮件服务器（172.16.16.231/cyberdyne.local 域）。

（3）远程邮件服务器接收邮件，并将其与适当的邮箱相关联。收件方计算机（172.16.16.235）上的邮件客户端通过 IMAP 协议取回邮件。

第 1 步：客户端至本地邮件服务器

我们从第一步开始逐步查看邮件发送过程，对应的数据包文件为 mail_sender_client_1.pcapng。这个文件从用户在邮件客户端上单击“发送”按钮开始，用户计算机和本地邮件服务器在前 3 个数据包中完成 TCP 握手。

注意　本节中，你可以忽略抓包文件中所有的“ETHERNET FRAME CHECK SEQUENCE INCORRECT”错误。在实验环境中生成这些数据包时的一些人工操作导致了这些错误。

连接建立后，客户端使用 SMTP 将邮件传输至邮件服务器。分析数据包时，你可以逐个浏览 SMTP 请求并回复数据包，在“数据包详情”窗口中查看 SMTP 部分，但是有一种更简便的方式。由于 SMTP 是一种简单的传输协议，并且在这个例子中通信过程为明文传输，因此你可以跟踪 TCP 流，从而在一个窗口中查看整个传输过程。在抓包文件中任意选择一个数据包，然后右键选择“Follow > TCP Stream”。传输的 TCP 流如图 9-30 所示。

异步社区Kensuke(18319066421) 专享 请尊重版权

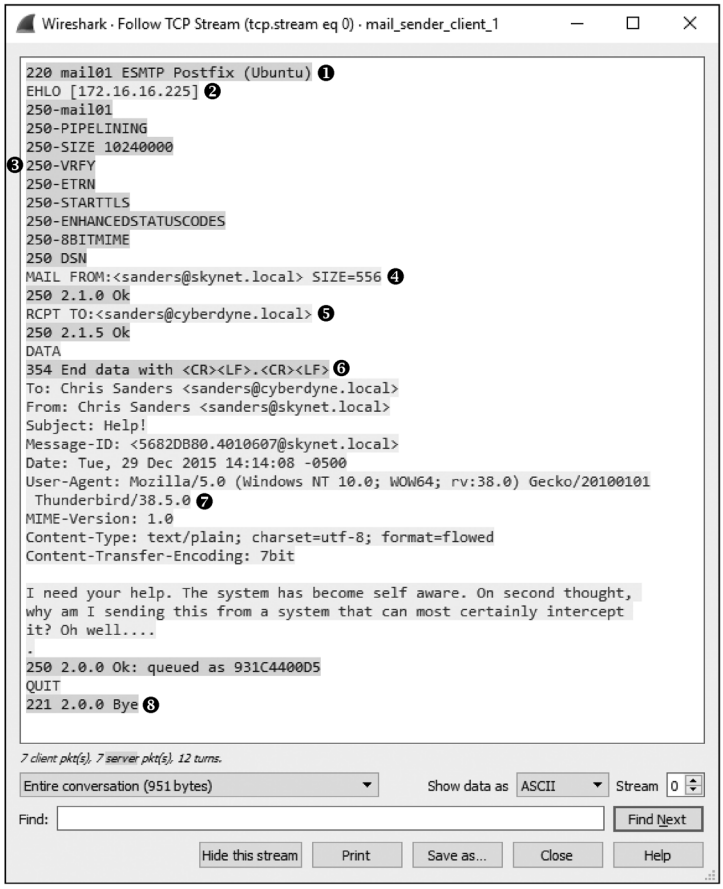


图 9-30　查看从邮件客户端至邮件服务器的 TCP 流

在连接建立后，邮件服务器在数据包 4 向客户端发送一个服务标签，表明此服务器已经准备好接收命令。在本例中，服务器显示为运行在 Ubuntu Linux 系统上的 Postfix 服务器❶。同时，此服务器支持接收扩展 SMTP（ESMTP）命令。ESMTP 是 SMTP 的扩展，允许在邮件传输中使用更多的命令。

邮件客户端在数据包 5 中通过 EHLO 命令进行回复❷。当邮件服务器支持 ESMTP 时，EHLO 作为“Hello”命令，让服务器识别发信的客户端主机。在不支持 ESMTP 的情况下，使用 HELO 命令进行发信客户端识别。在本例中，发信方使用 IP 地址作为识别方式，此外，域名也可以作为识别方式。

在数据包 7 中，服务器回复内容包括 VRFY、STARTTLS 和 SIZE 10240000❸。这些内容表示此 SMTP 服务器支持的命令，用于告知客户端在邮件传输过程中

异步社区Kensuke(18319066421) 专享 请尊重版权

可用的命令范围。在发送邮件之前，这个特征协商过程会发生在每次 SMTP 传输的开始部分。邮件传输从数据包 8 开始，这个抓包文件剩余的大部分内容均为邮件传输过程的数据包。

SMTP 由客户端发送的简单命令和参数值控制，服务器会回复相应的响应码。这种设计是为了协议简化，与 HTTP 和 TELNET 很相似。数据包 8 和数据包 9 是一组示例请求/回复，客户端发送了 MAIL 命令，参数为 FROM:<sanders@skynet.local> SIZE=556❹；服务器回复的响应码为 250（请求的邮件操作完成），参数为 2.1.0 Ok。在这次请求中，客户端发送了发信方的邮箱地址和邮件大小，服务器响应说明数据已经被接收，并且格式正确。类似的传输过程也发生在数据包 10 和数据包 11；客户端发送 RCPT 命令，参数为 TO:<sanders@cyberdyne.local>❺，服务器响应为 250 2.1.5 Ok。

剩余的部分是传输邮件内容数据。如数据包 12 所示，客户端使用 DATA 命令启动邮件内容传输过程。服务器响应码为 354，并附带信息❻，其中，响应码 354 表示服务器为这封邮件创建了缓冲区，客户端可以开始邮件传输；附带信息表示客户端需要发送一个<CR><LF>.<CR><LF>用于标记传输终止。这封邮件使用明文传输，并且响应码表示传输成功。你会注意到，邮件文本中包含一些附加信息，包括日期、内容类型和编码，以及传输使用的用户代理。以上信息表明，发信用户使用 Mozilla Thunderbird 发送了这封邮件❼。

传输完成后，在数据包 18 中，邮件客户端使用不带参数的 QUIT 命令，终止了 SMTP 连接。在数据包 19 中，服务器回复响应码 221（＜domain＞传输通道关闭），附带参数为 2.0.0 Bye❽。TCP 连接在数据包 20～23 中正常关闭。

第 2 步：本地服务器至远程服务器

接下来，我们将从 skynet.local 中的本地邮件服务器（地址为 172.16.16.221）的角度，考虑这个邮件传输场景。用于分析的数据包存储在 mail_sender_server_2.pcapng 文件中，这些数据包可以直接在邮件服务器上抓取。由于邮件从客户端至本地邮件服务器的传输过程，也被邮件服务器上的抓包工具记录，因此你会发现，mail2_sender_server_2.pcapng 包含的前 20 多个数据包与第 1 步中的数据包一致。

如果邮件目的地是 skynet.local 域中的其他邮箱，那么我们不会发现其他的 SMTP 传输；我们将看到某个邮件客户端使用 POP3 或 IMAP 协议收取邮件。然而，由于这封邮件被发送至 cyberdyne.local 域，因此本地 SMTP 服务器必须将邮件传输至为 cyberdyne.local 提供服务的远程邮件服务器。在数据包 22 中，本地邮件服务器 172.16.16.221 和远程邮件服务器 172.16.16.231 开始进行 TCP 握

异步社区Kensuke(18319066421) 专享 请尊重版权

手，并开始第 2 步传输过程。

注意

在现实场景中，邮件服务器通过一种特殊的 DNS 记录类型——邮件交换（MX）记录，定位其他的邮件服务器。由于本节的场景为实验室环境，远程邮件服务器的 IP 地址已经预配置在本地邮件服务器上，因此我们将不会看到 DNS 通信过程。在进行邮件发送故障检查时，需要考虑 DNS 问题及与邮件相关的特定协议的问题。

连接建立完成后，我们在数据包列表中看到，邮件传输至远程服务器的过程使用 SMTP 协议。跟踪传输过程的 TCP 流，能够更清晰地查看这个会话过程，如图 9-31 所示。如果需要将这个连接单独提取出来，请在过滤器中使用 tcp.stream==1 作为筛选条件。



图 9-31　查看从本地邮件服务器至远程邮件服务器的 TCP 流

异步社区Kensuke(18319066421) 专享 请尊重版权

这个传输过程与图 9-30 中的过程基本一致。本质上，两者传输的邮件内容一致，区别在于，步骤 2 中的传输过程发生在两个服务器之间，步骤 1 中的过程发生在客户端和服务器之间。远程服务器被标记为 mail02❶，本地服务器被标记为 mail01❷，两个服务器共享了一系列可用命令❸；邮件内容❹包括步骤 1 中完整的邮件文本，以及附加在 To: Chris Sanders <sander@cyberdyne.local> 行之前的部分。这个通信过程在数据包 27～35 中，以一个 TCP 断开作为结束。

从根本上说，服务器不关注邮件是来自于邮件客户端还是另一个 SMTP 服务器，所以客户端——服务器邮件传输中所有的规则和过程在服务器——服务器邮件传输过程中也都适用（任意形式的访问控制策略除外）。在实际情况中，本地邮件服务器和远程邮件服务器可能并不具备相同的特征参数集合（包括命令集和参数集），或是基于完全不同的平台。这也是 SMTP 要进行初始化通信的非常重要的原因；这一过程确保了在邮件传输之前，收件服务器将自身支持的命令和参数集发送给发件方，从而完成传输使用命令的协商。在 SMTP 客户端或服务器明确了收件服务器的特征参数后，能够保证发件方发送的 SMTP 命令被收件服务器识别，并且邮件被正常传输。这一机制使 SMTP 能够在大量不同的客户端和服务器之间应用，同时，也让我们在发送邮件时，不需要了解收件方的网络设备信息。

第 3 步：远程服务器至远程客户端

现在，我们的邮件到达了远程邮件服务器，此服务器负责将邮件投递至 cyberdyne.local 中的邮箱。我们将查看在远程邮件服务器上抓取的数据包 mail_receiver_server_3.pcapng，如图 9-32 所示。

我们再次发现前 15 个数据包很熟悉，因为传输的邮件内容与前两个步骤相同，不同的是，源地址变为本地邮件服务器❶，目的地址变为远程邮件服务器❷。这个流程完成后，SMTP 服务将邮件和指定的邮箱进行关联，之后，预期的收件方可以通过邮件客户端收取邮件。

正如之前提到的，SMTP 主要被用来发送邮件；截至目前，SMTP 也是发送邮件时常用的协议。从服务器上的邮箱收取邮件的方式更加多样，同时，由于在收取邮件事件中不断有新的需求，因此出现了数种用于完成邮件收取任务的协议。其中，比较流行的是邮局协议 v3（POP3）和 Internet 邮件访问协议（IMAP）。在本例中，远程客户端使用 IMAP 从邮件服务器收取邮件，对应通信发生在数据包 16～34 中。

异步社区Kensuke(18319066421) 专享 请尊重版权



图 9-32　查看从本地邮件服务器至远程邮件服务器的 TCP 流

本书并不会涵盖 IMAP 的内容；在本例中，即使我们介绍了 IMAP，也并不会对数据包分析带来多大帮助——这个通信过程是加密的。查看数据包 21❶，你会发现客户端（172.16.16.235）向邮件服务器（172.16.16.231）发送了 STARTTLS 命令，如图 9-33 所示。

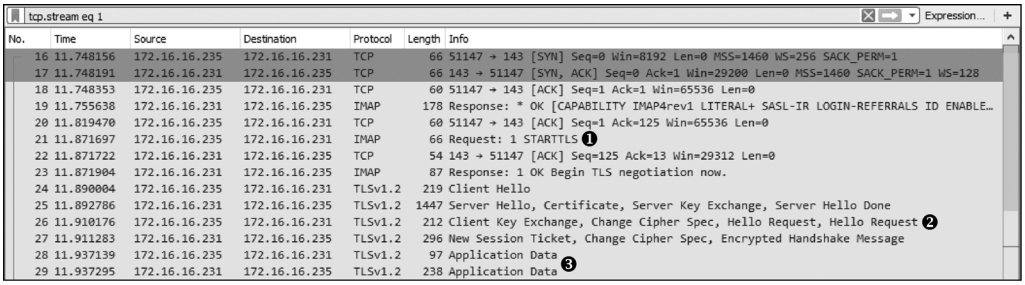


图 9-33　STARTTLS 命令表明此次 IMAP 通信将被加密

异步社区Kensuke(18319066421) 专享 请尊重版权

这个命令告知服务器，客户端将使用 TLS 加密收取邮件的过程。双方在数据包 24～27 中❷建立了安全信道，在余下的数据包中❸，邮件收取使用 TLS（传输层安全）协议完成。当查看这些数据包，或者尝试跟踪 TCP 流时（见图 9-34），你会发现内容不具备可读性，目的是防止邮件被恶意用户截获，避免数据劫持和嗅探的发生。

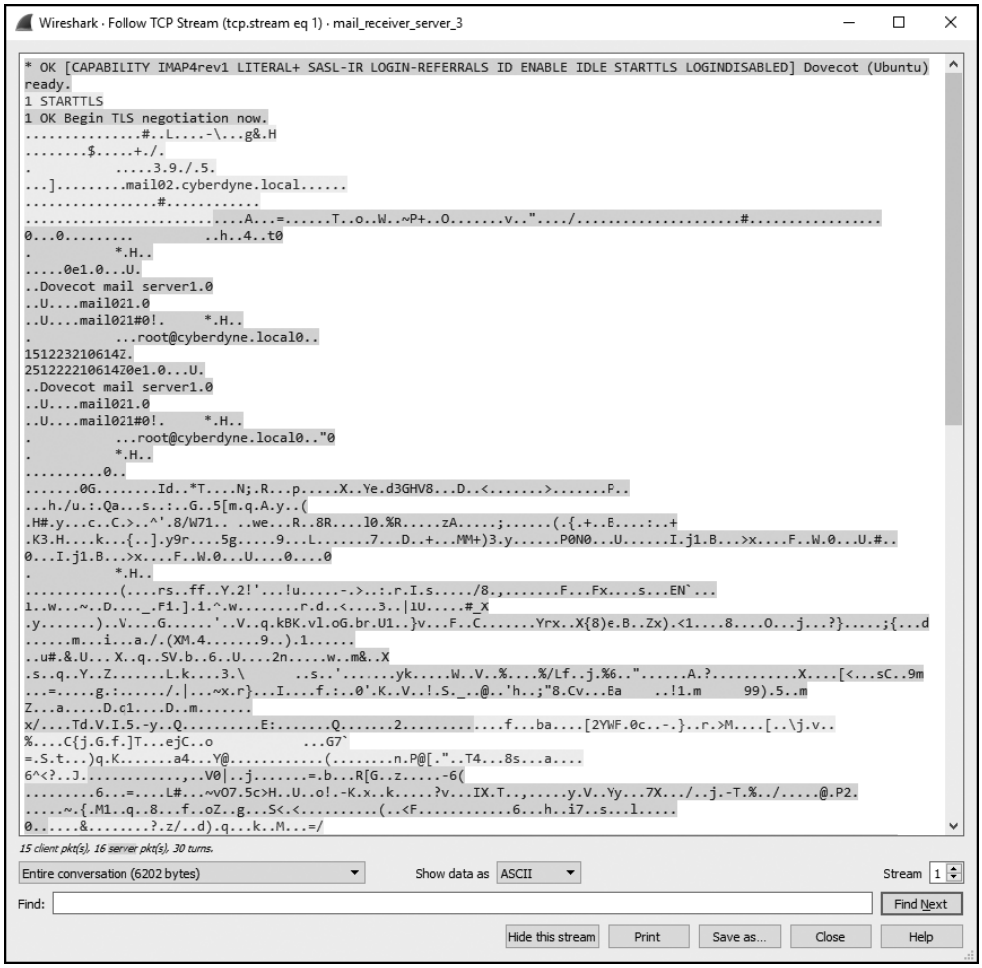


图 9-34　客户端下载邮件过程使用加密 IMAP 传输

最后，在这些数据包被接收后，不同域内两个用户之间的邮件发送过程就完成了。

异步社区Kensuke(18319066421) 专享 请尊重版权

9.4.3 使用 SMTP 发送附件

在设计 SMTP 计时从未计划使其成为一种传输文件的途径，但由于使用邮件发送文件的便捷性，因此它成为了很多人的首选文件共享方式。让我们使用一个简短的实例，从数据包层面分析 SMTP 的文件传输过程。

在抓取文件 mail_sender_attachment.pcapng 中，用户使用客户端（172.16.16.225）向同一网络内另一个用户发送邮件，本地 SMTP 服务器位于 172.16.16.221。这封邮件包含一些文本内容，以及一个图片文件附件。

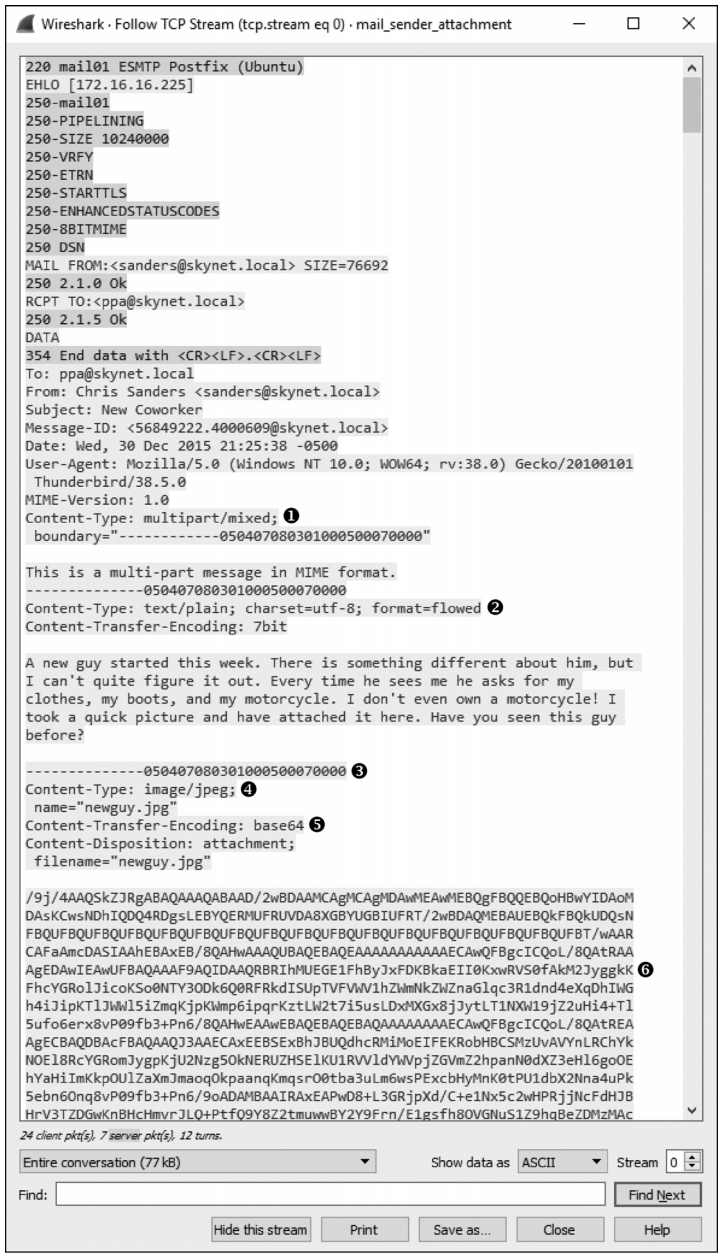
使用 SMTP 发送附件与发送文本没有太多区别。它们都只是向服务器发送数据；尽管过程中经常会使用一些特殊编码，我们仍使用 DATA 命令进行数据传送。请打开抓包文件，跟踪 SMTP 传输的 TCP 流，查看这一操作过程。TCP 流如图 9-35 所示。

本例的通信过程在开始部分与之前的场景类似，包括服务识别和可用协议信息交换。当客户端准备传输邮件信息时，它会提供发件方地址和收件方地址，并发送 `DATA` 命令，通知服务器分配用于接收邮件数据的缓冲区。从这部分开始出现了少许差异。

在之前的例子中，客户端将文本直接传输至服务器，然后传输完成。在本例中，除了明文文本信息之外，客户端还需要发送图片附件的二进制数据。为了实现这个目的，客户端将内容类型标记为 `multipart/mixed`，以 `------------050407080301000500070000`❶作为文本信息和二进制数据的分界线。这告知服务器，传输的邮件内容包含多种类型的数据，每种数据类型有特定的 MIME 类型和编码方式，各种类型的数据使用指定的边界值分隔。通过这种机制，当另一个邮件客户端接收邮件时，基于分界线和每个数据块内指定的 MIME 类型、编码方式，接收端能够知道如何解析邮件数据。

在本例中，邮件数据包含两部分。第一部分是邮件文本，内容类型为 text/plain❷。在此之后，我们能看到一个分隔标记和第二部分的起始❸。第二部分包含图片文件，内容类型为 `image/jpeg`❹。同样值得注意的是，`Content-Transfer-Encoding` 值设为 `base64`❺，这表示数据需要使用 base 64 解码。余下的数据包中包含编码过的图片文件❻。

异步社区Kensuke(18319066421) 专享 请尊重版权


Wireshark · Follow TCP Stream (tcp.stream eq 0) · mail_sender_attachment
220 mail01 ESMTP Postfix (Ubuntu)
EHLO [172.16.16.225]
250-mail01
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
MAIL FROM:<sanders@skynet.local> SIZE=76692
250 2.1.0 Ok
RCPT TO:<ppa@skynet.local>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
To: ppa@skynet.local
From: Chris Sanders <sanders@skynet.local>
Subject: New Coworker
Message-ID: <56849222.4000609@skynet.local>
Date: Wed, 30 Dec 2015 21:25:38 -0500
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:38.0) Gecko/20100101
Thunderbird/38.5.0
MIME-Version: 1.0
Content-Type: multipart/mixed; ❶
boundary="------------050407080301000500070000"
This is a multi-part message in MIME format.
--------------050407080301000500070000
Content-Type: text/plain; charset=utf-8; format=flowed ❷
Content-Transfer-Encoding: 7bit
A new guy started this week. There is something different about him, but
I can't quite figure it out. Every time he sees me he asks for my
clothes, my boots, and my motorcycle. I don't even own a motorcycle! I
took a quick picture and have attached it here. Have you seen this guy
before?
--------------050407080301000500070000 ❸
Content-Type: image/jpeg; ❹
name="newguy.jpg"
Content-Transfer-Encoding: base64 ❺
Content-Disposition: attachment;
filename="newguy.jpg"
/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAMCAgMCAgMDAwMEAwMEBQgFBQQEBQoHBwYIDAoM
DAsKCwsNDhIQDQ4RDgsLEBYQERMUFRUVDA8XGBYUGBIUFRT/2wBDAQMEBAUEBQkFBQkUDQsN
FBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBT/wAARR
CAFaAmcDASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAA
AgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkK
FhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWG
h4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl
5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREA
AgECBAQDBAcFBAAAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYk
NOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOE
hYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk
5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwD8+LJQ9Y8Frn/E1sh5Z2jGfbHOPrXpHhnw
HrPiqzuLnTLcTQwOEcmRVwSMjqRnpXPeJ/DWqeFtQNnqlsYJSNyHIZXHqCOtfnuIwtWNJ1Gt
Ln7Jg8dQlXVNPU5Wiiiuc9QKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA
KKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACii
igAooooAKKKKACiiigAooooA

瞬间解码，任何截获这一通信的攻击者都能够毫不费力地获得此图片文件。如果你想要自行将图片文件从抓包文件中分离出来，在第 12 章的远程接入木马部分，有一个类似的场景——从基于 HTTP 的文件传输中分离一个图片；阅读了该章节后，回到本例的数据包分析过程，你可以尝试找出这个发件人的神秘的新同事是谁。

9.5 小结

这一章介绍了你在检查应用层流量时会遇到的常见协议。在下面的一章中，我们将通过探索大量实战场景，来介绍一些新的协议，以及我们这里所介绍协议的额外功能。

如果希望学习更多关于协议的知识，可以阅读它们相关的 RFC 文档，或者看一看 Charles Kozeriok 的 *The TCP/IP Guide*(No Starch Press, 2005)。同样也可以看一下本书附录中列出的资源。

异步社区Kensuke(18319066421) 专享 请尊重版权

第10章

基础的现实世界场景



从本章开始，我们将深入到数据包分析的内涵，使用Wireshark分析现实世界中的网络问题。在本章第一部分，我们将分析网络工程师、服务台技术人员、应用开发者在日常工作中会遇到的场景——全都来自于我与同事们的实际经验。我们将使用Wireshark，查看来自Twitter、Facebook和ESPN的流量，观察这些常用的服务是如何工作的。

本章第二部分将介绍一系列实际问题。针对每一个具体问题，我描述了它们的情况，并把当时可用的信息提供给分析者。在这个基础上，转到分析数据包的过程、描述捕获特定数据包的方法，以及分析过程的每个步骤。分析完成后，我将提供一个完整的问题解决方案，或是指出可能的解决方法，并总结从中汲取的经验教训。

自始至终，要记住分析是一个非常动态的过程，并且，我用来分析每一个场景的方法可能跟你用的不一样。每个人可以用不同的方法来分析。但最重要

异步社区Kensuke(18319066421) 专享 请尊重版权

的是，分析的最终结果能够解决一个问题，或使你获得学习经验。另外，本章讨论的大部分问题，即使不用数据包嗅探器，也可以解决。当我初次学习分析数据包时，发现反常规地使用数据包分析技术来查看典型问题有很多好处，这也是我给你介绍这些场景的原因。

10.1 丢失的网页内容

在第一个场景中，我们的用户是 Packet Pete，一位从不熬夜，导致经常错过西海岸赛事的大学篮球迷。每天上午，Pete 打开计算机的第一件事是访问 ESPN，查看前一天晚上的最终比分。

这个上午，当 Pete 浏览 ESPN 时，他发现加载网页耗费了很长时间；加载过程终于完成时，页面丢失了大部分图片和内容（见图 10-1）。让我们帮助 Pete 诊断这个问题。

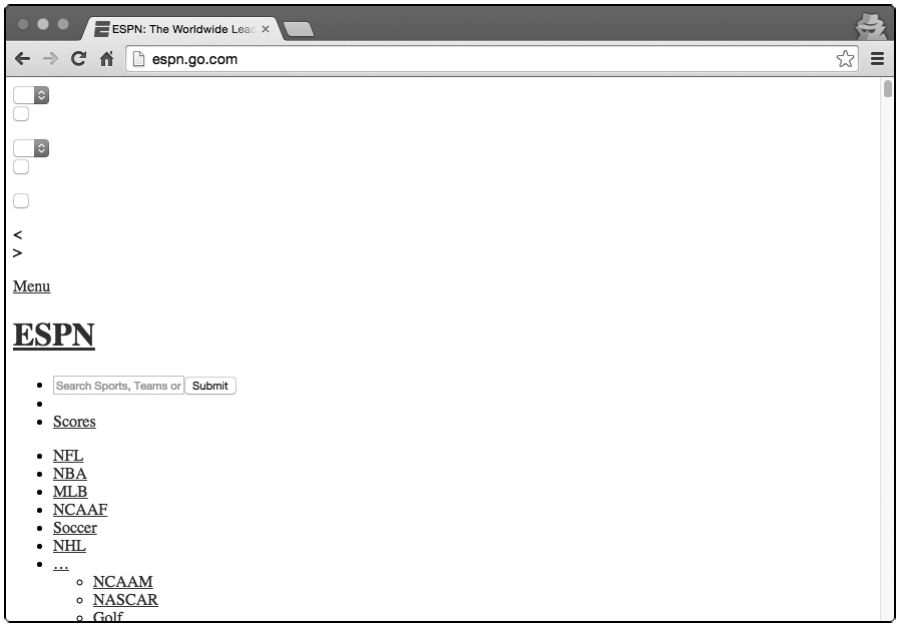


图 10-1 ESPN 页面加载失败

10.1.1 侦听线路

这个问题只在 Pete 的计算机上发生，并没有影响到其他人，因此我们在他

异步社区Kensuke(18319066421) 专享 请尊重版权

的计算机上进行抓包工作。为此，我们将安装 Wireshark，抓取浏览 ESPN 网站的流量。这些数据包位于 http_espn_fail.pcapng 文件中。

10.1.2 分析

我们知道，Pete 的问题在于，他不能浏览某一个网站，所以我们首先考虑 HTTP 协议。如果你阅读了之前的章节，那么你会对客户端和服务器之间的 HTTP 通信有基本的理解。我们建议，从客户端向远程服务器发送 HTTP 请求的地方开始分析。你可以使用过滤器筛选 GET 请求数据包（使用 `http.request.method == "GET"`），也可以使用主选单中的 **Statics > HTTP > Requests**（见图 10-2）进行过滤。

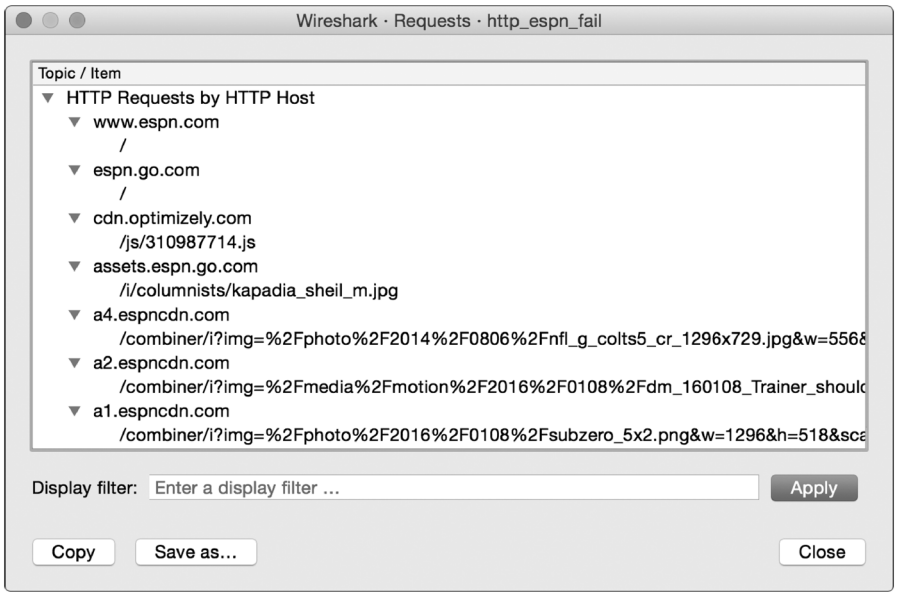


图 10-2　查看访问 ESPN 网站的 HTTP 请求

从这个概览中看到，抓取的数据包只包含 7 次不同的 HTTP 请求，这些请求都与 ESPN 网站有关。除了一个为大量网站投放广告的内容分发网络（CDN）服务，每个请求都在域名部分包含字符串 `espn`。在浏览含有广告或其他外部内容的网站时，通常会看到大量对 CDN 服务的请求。

在没有明确目标会话的情况下，下一个步骤是，选择 **Statistics > Protocol Hierachy** 对抓包文件进行协议分层统计。这将帮助我们定位非预期的协议和通

异步社区Kensuke(18319066421) 专享 请尊重版权

信过程中各种协议分布情况的异常（见图 10-3）。请注意，协议分层基于当前的筛选条件进行。为了对整个抓包文件进行分析以获得期望的结果，请先清除之前的过滤器设置。

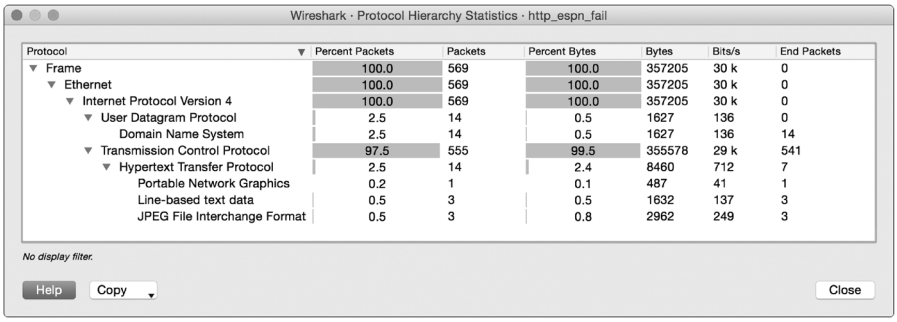


图 10-3　查看网页浏览会话的协议分层统计

协议分层视图并不复杂，我们能够很快地辨别出，会话只涉及两个应用层协议：HTTP 和 DNS。正如在第 9 章中学习到的，DNS 用于将域名转换为 IP 地址。因此，当访问一个网站，如 ESPN 时，如果你的系统没有缓存这个站点的 DNS 记录，则系统将发送一个 DNS 请求来查询远程网站服务器的 IP 地址。当 DNS 服务返回了一个可用的 IP 地址时，则域名解析信息将被添加到本地缓存中。此时，HTTP 通信（使用 TCP 协议）可以开始了。

虽然分层统计结果看上去并没有什么异常，但是 14 个 DNS 数据包是值得注意的。一个查询单个域名的 DNS 请求通常只包含一个数据包，请求响应同样由一个数据包构成（除非响应数据包很大，这种情况下，DNS 数据包将使用 TCP 传输）。会话中包含 14 个 DNS 数据包，这说明可能发生了多达 7 次的 DNS 请求（7 个请求数据包 + 7 个响应数据包 = 14 个数据包）。如图 10-2 所示，Pete 向 7 个不同的域名发起了请求，但是他在浏览器中只输入了一个 URL。其他请求是如何产生的呢？

在理想状况下，浏览网页只需要查询到一个服务器地址，然后在一次 HTTP 会话中即可获取所有的内容。但在实际情况中，一个网页可能提供多个服务器上的内容。可能的场景是，全部基于文本的内容在一处，图像内容在另一处，内嵌的视频则在第三个位置。这还不包括广告，广告可能由位于很多独立服务器上的多个提供商发放。当 HTTP 客户端解析 HTML 代码时，发现了对其他服务器上资源的引用，为了获取引用的内容，客户端将尝试查询相关的服务器，

异步社区Kensuke(18319066421) 专享 请尊重版权

这导致了额外的 DNS 查询和 HTTP 请求。这就是在 Pete 访问 ESPN 时发生的事情。在他想要查看来自于单个源的内容时，HTML 代码中引用了其他源的内容，之后，他的浏览器自动从其他多个域名请求内容。

明白了这些额外的请求是如何产生的，接下来，我们将逐个检查与每个请求相关的会话（**Statistics > Conversations**）。图 10-4 所示的会话窗口为我们提供了重要线索。

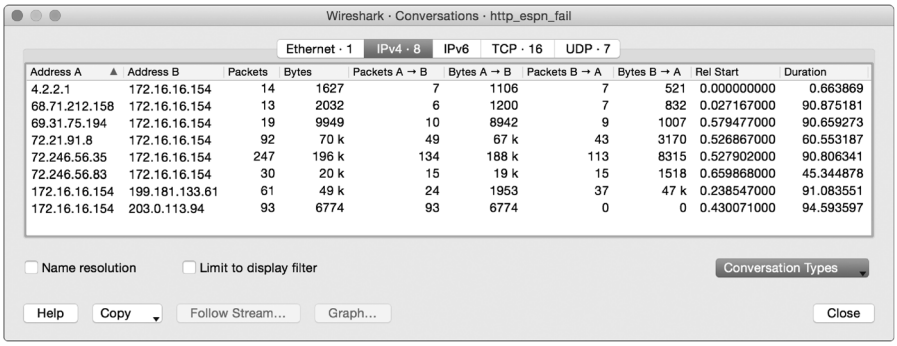


图 10-4　查看 IP 间会话

我们之前发现浏览器进行了 7 次 DNS 请求，以及相应的 7 次 HTTP 请求。在这个前提下，我们可以推断，有 7 个对应的 IP 间会话，然而，实际情况并非如此。这里出现了 8 个会话。应该如何解释这个情况？

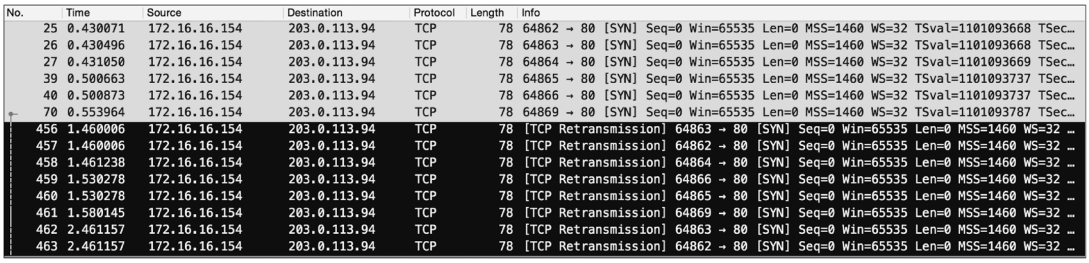
一种可能性是，抓包文件被一个与本例中问题无关的会话“污染”。确保分析工作不被无关的通信扰乱是很重要的，但是这并不是这个会话中出现问题的原因。如果仔细检查每一个 HTTP 请求，并留意请求的目的 IP 地址，你会发现存在一个会话，没有对应的 HTTP 请求。这个会话的双方为 Pete 的计算机（172.16.16.154）和远程 IP 203.0.113.94。这个会话在图 10-4 的最后一行被列出。我们注意到，Pete 的计算机发送了 6774 字节至未知服务器（203.0.113.94），但是收到了 0 字节，这值得我们去深究。

现在，筛选并查看这个会话（在会话上右击，并选择 **Apply As Filter > Selected > A<->B**），然后，运用 TCP 的知识来确定错误位置。

在正常的 TCP 通信中，我们期望看到一个标准的 SYNSYN/ACK-ACK 握手序列。在本例中，Pete 的计算机向 203.0.113.94 发送了一个 SYN 数据包（见图 10-5），但是我们没有看到 SYN/ACK 响应。不仅如此，Pete 的计算机还向不可用的目的服务器发送了多个 SYN 数据包，最终导致了他的机器发送 TCP 重

异步社区Kensuke(18319066421) 专享 请尊重版权

传数据包。我们将在第 11 章中具体讨论 TCP 重传，此处的关键是，一个主机发送了数据包，但是从未接收到响应。查看时间栏，我们发现，重传持续了 95s，并且没有任何响应。在网络通信中，这堪称龟速。



No.	Time	Source	Destination	Protocol	Length	Info
25	0.430071	172.16.16.154	203.0.113.94	TCP	78	64862 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093668 TSec…
26	0.430496	172.16.16.154	203.0.113.94	TCP	78	64863 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093668 TSec…
27	0.431050	172.16.16.154	203.0.113.94	TCP	78	64864 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093669 TSec…
39	0.500663	172.16.16.154	203.0.113.94	TCP	78	64865 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093737 TSec…
40	0.500873	172.16.16.154	203.0.113.94	TCP	78	64866 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093737 TSec…
70	0.553964	172.16.16.154	203.0.113.94	TCP	78	64869 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093787 TSec…
456	1.460006	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64863 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …
457	1.460006	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64862 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …
458	1.461238	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64864 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …
459	1.530278	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64866 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …
460	1.530278	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64865 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …
461	1.580145	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64869 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …
462	2.461157	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64863 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …
463	2.461157	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64862 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 …

图 10-5　查看非预期的连接

我们识别了 7 次 DNS 请求、7 次 HTTP 请求和 8 个 IP 间会话。在确定抓包文件并没有被额外数据污染的情况下，我们有理由相信，神秘的第 8 个 IP 间会话可能是 Pete 加载网页缓慢并最终失败的原因。由于某些原因，Pete 的计算机试图与一个不存在或未提供服务的设备进行通信。为了了解这个情况出现的原因，我们不会查看抓包文件中的数据包；相反，我们会考虑文件之外的内容。

当 Pete 浏览 ESPN 网页时，他的浏览器识别了其他域名上的内容。为了获取这些内容数据，他的计算机发起 DNS 查询来获得这些域名的 IP 地址，之后，向这些目标 IP 发起 TCP 连接，从而发送 HTTP 请求并获取网页内容。对于与 203.0.113.94 的会话，并没有对应的 DNS 请求。那么，Pete 的计算机是怎样获得这个地址的呢？

如果你记得我们在第 9 章中关于 DNS 的讨论，或是熟悉 DNS，那么你就会明白，大部分系统使用一种 DNS 缓存机制。这一机制在系统内建立本地的域名与 IP 地址间的映射缓存，当访问已经缓存 DNS 记录、经常访问的域名时，系统将使用本地的映射缓存，而不发起 DNS 请求。直至这些域名与 IP 地址间的映射过期，才进行新的 DNS 请求以获得域名的 IP 地址。然而，如果域名与 IP 的映射关系变更了，但是某个设备并没有发起新的 DNS 请求以获得新的地址，那么，在下次访问这一域名时，该设备将尝试连接一个无效的地址。

在 Pete 的例子中，就出现了这种情况。Pete 的计算机缓存了某一个为 ESPN

异步社区Kensuke(18319066421) 专享 请尊重版权

网站提供内容的主机的域名解析信息。由于缓存信息的存在，因此获取页面内容时没有进行 DNS 请求，这导致他的系统试图继续连接旧的地址。然而，这个地址已经不再为提供 HTTP 服务。作为结果，请求超时，并且内容加载失败。

对于 Pete 而言，值得庆幸的是，手动清除 DNS 缓存，只需要在命令行中敲击很短的命令即可。或者，他可以等待几分钟，在 DNS 缓存记录过期、新的 DNS 请求完成后，再尝试访问 ESPN 网站。

10.1.3 学到的知识

为了获得“肯塔基大学以 90 分击败了杜克大学”的消息，我们做了很多工作，但是这一过程使我们对网络主机间的关系有了更深层次的理解。在这个场景中，我们通过查看与请求的多个相关数据提供点以及出现在抓包文件中的会话，定位了问题并得出了解决方案。我们从定位到通信中少许的不一致信息入手，发现了客户端与 ESPN 的一个 CDN 服务器之间连接不可用的问题。

浏览一系列数据包以发现一些看起来可笑的数据，这样简单的方式不太适用于实际的问题诊断工作。即使是处理最简单的问题，也需要使用 Wireshark 的分析和统计特性对大量的抓包文件进行处理，以定位异常事件。熟练掌握这样的分析方式，对于在数据包层面顺利完成故障处理很重要。

如果想了解，正常情况下浏览器和 ESPN 之间的通信过程是怎样的，你可以使用浏览器访问这个网站，同时进行抓包，并尝试识别出所有负责内容分发的服务器。

10.2 无响应的气象服务

第二个场景还是和我们的朋友 Packet Pete 有关。他有众多爱好，包括气象学——他自认为是业余气象学家，每隔几小时就会查看天气现状和天气预报。他并不只是依赖于本地新闻的天气预报；他还在自己家的外面运营了一个小型气象站，并将报告数据上传至 WunderGround，用于数据汇总和查阅。今天，Pete 检查气象站，以查看昨晚的降温情况，结果发现，自午夜开始，气象站超过 9 小时未向 WunderGround 发送报告（见图 10-6）。

异步社区Kensuke(18319066421) 专享 请尊重版权

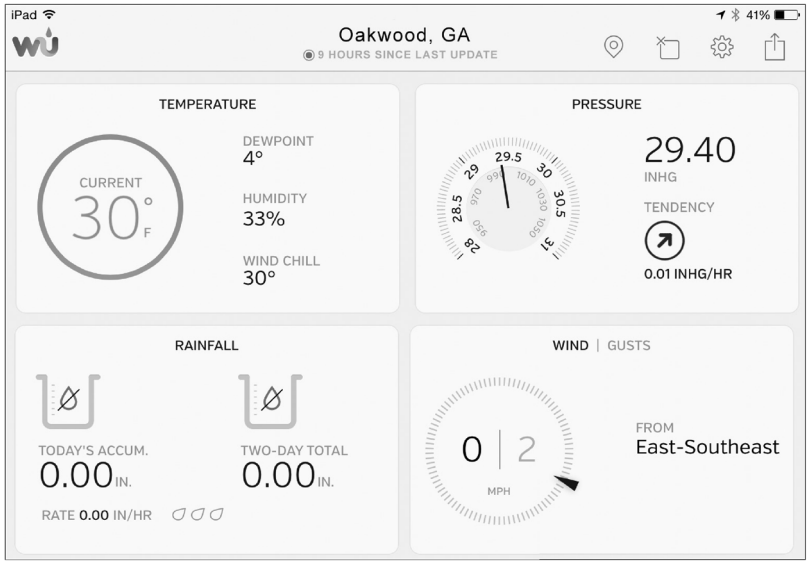


图 10-6　气象站在 9 小时内未发送报告

10.2.1　侦听线路

在 Pete 的网络中，气象站被挂在房顶上，与屋内的接收器通过射频连接。接收器通过路由器接入网络，然后将统计数据发送至互联网上的 WunderGround 服务器。这个结构如图 10-7 所示。

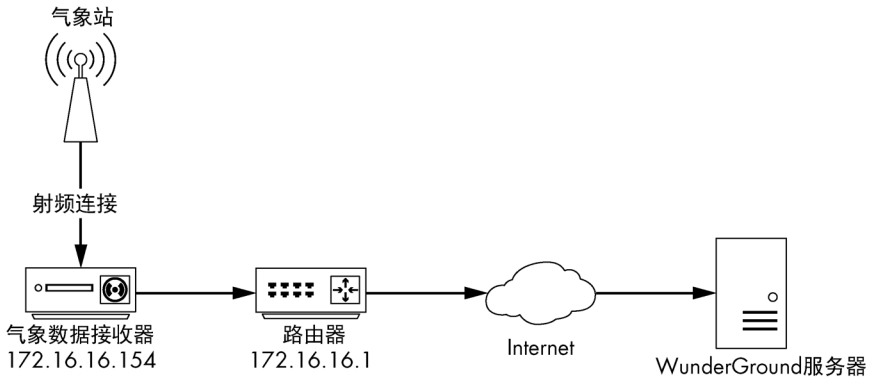


图 10-7　气象站网络结构

接收器有一个简易的 Web 管理界面，当 Pete 登录管理界面时，只发现了一条与上次数据同步时间有关的加密消息，除此之外，没有其他可用于故障处理

异步社区Kensuke(18319066421) 专享 请尊重版权

的信息——这个软件不提供任何详细的错误日志。由于接收器是气象站设备的通信枢纽，因此抓取发往或来自于接收器的数据包对于诊断问题是有意义的。由于这是一个家庭网络，因此家用路由器可能不支持端口镜像功能。我们最好使用廉价的侦听器或使用 ARP 缓存投毒来截获数据包。抓取的数据包在文件 weather_broken.pcapng 中。

10.2.2 分析

打开抓包文件后，你会看到，这又是一个 HTTP 通信的问题。抓取的数据包限定于 Pete 的本地气象数据接收器 172.16.16.154 与互联网上一个未知的远程设备 38.102.136.125 之间的单个会话中（见图 10-8）。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.16.16.154	38.102.136.125	TCP	78	53904 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1015238041 TSecr=0 SACK_PERM=1
2	0.087018	38.102.136.125	172.16.16.154	TCP	60	80 → 53904 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1350
3	0.087108	172.16.16.154	38.102.136.125	TCP	54	53904 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
4	0.087178	172.16.16.154	38.102.136.125	HTTP	571	GET /weatherstation/updateweatherstation.php?ID=KGAOAKWO2&PASSWORD=00000000&tempf=43.0&humidity=30…
5	0.176462	38.102.136.125	172.16.16.154	HTTP	237	HTTP/1.0 200 OK (text/html)
6	0.176567	172.16.16.154	38.102.136.125	TCP	54	53904 → 80 [ACK] Seq=518 Ack=184 Win=65535 Len=0
7	0.176714	172.16.16.154	38.102.136.125	TCP	54	53904 → 80 [FIN, ACK] Seq=518 Ack=184 Win=65535 Len=0
8	0.262587	38.102.136.125	172.16.16.154	TCP	60	80 → 53904 [FIN, ACK] Seq=184 Ack=519 Win=7673 Len=0
9	0.262656	172.16.16.154	38.102.136.125	TCP	54	53904 → 80 [ACK] Seq=519 Ack=185 Win=65535 Len=0

图 10-8 分离出的气象站接收器通信

在检查这个会话的数据之前，让我们先来识别这个未知 IP。如果不进行进一步研究，我们将无法判断这个 IP 是否是 Pete 的气象站应该访问的地址，但是我们至少能够通过 WHOIS 查询来确定此 IP 是否是 Wunderground 服务器的一部分。你可以在大多数域名注册网站或区域互联网注册管理网站完成 WHOIS 查询。根据查询结果，这个 IP 看起来属于一家名为 Cogent 的互联网服务供应商（ISP）（见图 10-9）。结果中也提到了 PSINet 公司，但是搜索显示，21 世纪初 Cogent 获得了 PSINet 的大部分设备。

在某些情况下，如果 IP 地址由一个组织或企业直接注册，那么 WHOIS 查询会返回组织名称。然而，多数情况下，公司不会自己去直接申请 IP，而是从因特网服务提供商（ISP）的 IP 池中获取地址。在这种情况下，另一种有效的措施是查找与 IP 地址相关联的自主系统编号（ASN）。组织需要申请一个 ASN 用于在公网中支持某些路由方式。有很多方法可用于查找 IP-ASN 关联关系（一些 WHOIS 查询工具会自动查找 IP-ASN 关联），我推荐使用 Cymru 团队自动查询工具。使用这个工具查询 38.102.136.125，我们看到它与 AS 36347 相关联，此 ASN 属于“WUNDERGROUND – THE WEATHER CHANNEL, LLC, US”（见图 10-10）。这表明，至少，与气象站进行通信的设备属于期望中的组织。如果查询结果返回的组织信息与期望值不符，则说明 Pete 的接收器通信对象设备可

异步社区Kensuke(18319066421) 专享 请尊重版权

能有误，但是在本例中并没有出现这种情况。

Network	
Net Range	38.0.0.0 - 38.255.255.255
CIDR	38.0.0.0/8
Name	COGENT-A
Handle	NET-38-0-0-0-1
Parent	
Net Type	Direct Allocation
Origin AS	AS174
Organization	PSINet, Inc. (PSI)
Registration Date	1991-04-16
Last Updated	2011-05-20
Comments	Reassignment information for this block can be found at rwhois.cogentco.com 4321
RESTful Link	https://whois.arin.net/rest/net/NET-38-0-0-0-1

Function	Point of Contact
Tech	PSI-NISC-ARIN (PSI-NISC-ARIN)

See Also	Related organization's POC records.
See Also	Related delegations.

图 10-9　WHOIS 数据识别出此 IP 的拥有者

Executing commands. Please be patient!

v4.whois.cymru.com

The server returned 4 line(s).

```
[Querying v4.whois.cymru.com]
[v4.whois.cymru.com]
AS      | IP               | AS Name
36347   | 38.102.136.125   | WUNDERGROUND - THE WEATHER CHANNEL, LLC,US
```

图 10-10　对这个外网 IP 地址进行 IP-ASN 关联查询

确认了这个未知主机的所属组织信息后，我们将深入探讨通信的细节。这个会话相对较短：由一个 TCP 握手过程、一次 HTTP `GET` 请求及响应和一个 TCP 断开过程组成。TCP 握手和断开似乎成功了，所以问题可能出现在 HTTP 请求中。我们跟踪 HTTP 请求的 TCP 流进行细致查看（见图 10-11）。

在 HTTP 通信过程中，首先由 Pete 的接收器向 WunderGround 发送一个 `GET` 请求。HTTP 内容部分没有数据，大量的数据通过 URL 进行传输❶。对于 Web 应用，通过 URL 查询字符串传输数据是很常见的，看起来，接收器使用这种机制更新天气信息。例如，你可以看到 `tempf=43.0`、`dewptf=13.6` 和 `windchllf=43.0`

异步社区Kensuke(18319066421) 专享 请尊重版权

这样的字段。WunderGround 信息收集服务器解析 URL 中的一些字段和参数，并将它们存储在数据库中。



图 10-11　跟踪接收器通信的 TCP 流

根据第一印象，这个发往 WunderGround 服务器的 `GET` 请求似乎没有任何问题，但是对应的响应显示有一个错误。服务器的响应状态码为 `HTTP/1.0 200 OK`❷，表明 GET 请求被成功接收，但是响应的消息体包含了一条有用的信息，`INVALIDPASSWORD|Password or key add/or id are incorrect`❸。

回到请求 URL 部分，你会发现查询字符串的前两个参数为 `ID` 和 `PASSWORD`。气象站使用这种方式完成在 WunderGround 服务器上的登录和验证。

本例中，Pete 的气象站 ID 正确，但是密码错误。由于某些未知原因，密码被置为 0。由于已知的最后一次通信成功发生在午夜，因此可能是一次升级或接收器重启，导致密码设置丢失。

注意　由于很多开发者选择使用 URL 传递参数，因此我们建议一般情况下不要如本例所示，将密码写在 URL 参数中。因为在不使用加密措施，如 HTTPS，的情况下，HTTP 通信将使用明文传输请求的 URL。所以，在使用 URL 参数传递密码时，碰巧正在监听通信链路的恶意用户能够截获你的密码。

此刻，Pete 接入他的接收器，输入新的密码。稍后，他的气象站重新开始同步数据。一个成功的气象站通信数据在 weather_working.pcapng 中。通信流如图 10-12 所示。

异步社区Kensuke(18319066421) 专享 请尊重版权

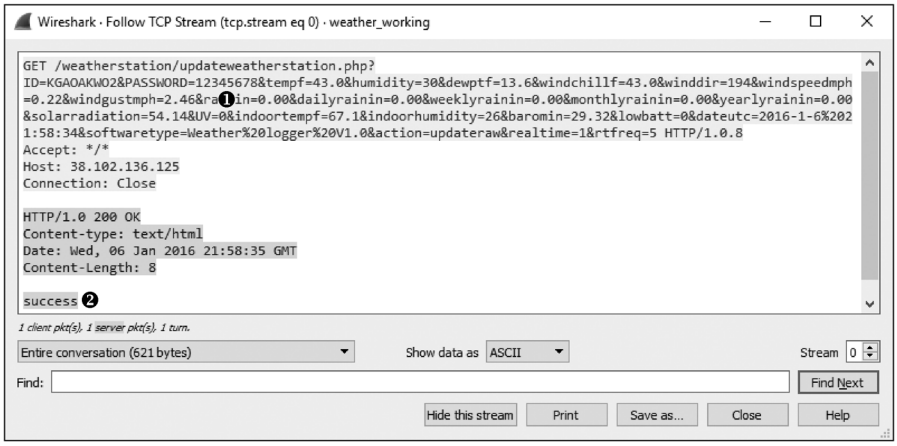


图 10-12　成功的气象站通信

现在，密码正确❶，WunderGround 服务器在应答的 HTTP 响应体中返回了一条 `success` 消息❷。

10.2.3　学到的知识

在这个场景中，一个第三方服务使用另一个协议（HTTP）的特性完成网络访问。你会经常需要修复第三方服务的网络问题，当没有相应的文档或错误日志时，数据包分析技术非常适合用于对这些服务进行故障处理。现在物联网（IoT）设备，如这个气象站，在生活中的应用越来越多，本节中的情况将越来越普遍。

解决这样的问题，需要有能力检查未知的传输序列，推断出通信过程可能的工作方式。某些应用，如本例中基于 HTTP 的气象数据传输，相当简单。一些其他的应用非常复杂，包括多种传输方式、数据加密，甚至是 Wireshark 原生组件无法解析的定制化协议。

随着对更多的第三方服务的研究，你会逐渐了解开发者用于实现网络通信的常见模式。这样的知识会提高你对这些服务进行故障处理时的效率。

10.3　无法访问 Internet

在许多情况下，你可能需要诊断和解决 Internet 的连接问题，下面我们将介

异步社区Kensuke(18319066421) 专享 请尊重版权

绍一些你可能遇到的常见问题。

10.3.1 网关配置问题

第一个问题的场景相当简单：用户不能访问 Internet。我们已经确认该用户可以访问所有内网资源，包括其他工作站的共享内容以及运行在本地服务器上的应用程序。

这个网络架构非常简单，因为所有客户机和服务器都连接到一系列的简单交换机上。Internet 连接由一个路由器（作为默认网关）处理，IP 地址信息由 DHCP 服务提供。这种情景在小型办公室中非常常见。

1. 侦听线路

为了找出问题的原因，我们可以一边用嗅探器监听线路，一边让用户尝试浏览 Internet。我们使用 2.5 节中的信息（见图 2-15），来决定放置嗅探器的方法。

网络上的交换机不支持端口镜像。为了完成测试，我们已经不可避免地妨碍了用户，所以我们假设可以使他再次下线（这是说，使用网络分流器是监听线路的推荐办法）。最后得到捕获记录文件 nowebaccess1.pcap。

2. 分析

如图 10-13 所示，流量捕获记录文件以一个 ARP 请求和响应开始。用户计算机的 MAC 地址是 00:05:b3:bf:91:ee，IP 地址是 172.16.0.8。在数据包 1 中，用户的计算机发送一个 ARP 广播数据包给网络上的所有计算机，试图得到默认网关 172.16.0.10 的 MAC 地址。

No.	Time	Source	Destination	Protocol	Length	Info
1	04:32:21.445645	00:25:b3:bf:91:ee	ff:ff:ff:ff:ff:ff	ARP	42	Who has 172.16.0.10? Tell 172.16.0.8
2	04:32:21.445735	00:24:81:a1:f6:79	00:25:b3:bf:91:ee	ARP	60	172.16.0.10 is at 00:24:81:a1:f6:79

图 10-13　针对计算机默认网关的 ARP 请求和响应

根据数据包 2 中收到的响应，用户的计算机了解到 172.16.0.10 的 MAC 地址是 00:24:81:a1:f6:79。收到这个响应后，计算机就有了到达网关的路由，而网关应该可以带它接入 Internet。

ARP 响应之后，计算机会在数据包 3 中请求将网站的域名解析成 IP 地址。如图 10-14 所示，计算机发送一个 DNS 查询数据包到它的首选 DNS 服务器 4.2.2.2❶。

异步社区Kensuke(18319066421) 专享 请尊重版权

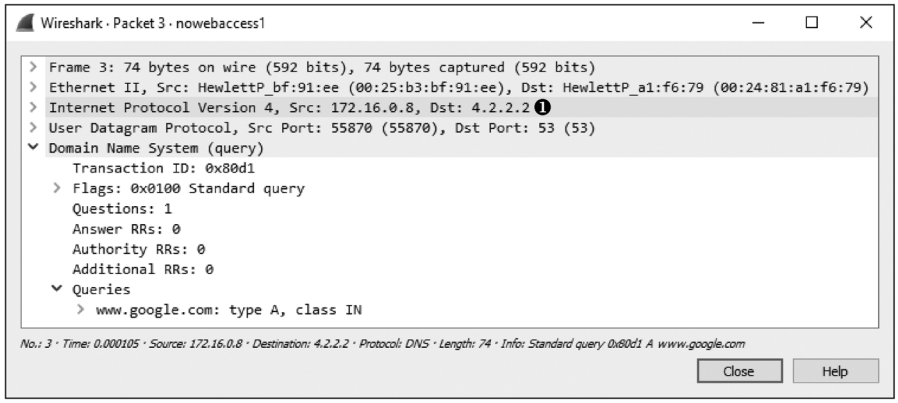


图 10-14　发送到 4.2.2.2 的 DNS 查询

正常情况下，DNS 服务器会迅速响应 DNS 查询，但在这个例子中并非如此。我们没有看到任何响应，却发现同样的 DNS 查询再次发送到不同的目的地址。如图 10-15 所示，在数据包 4 中，第二个 DNS 查询被发送到预先配置好的备用 DNS 服务器 4.2.2.1❶。

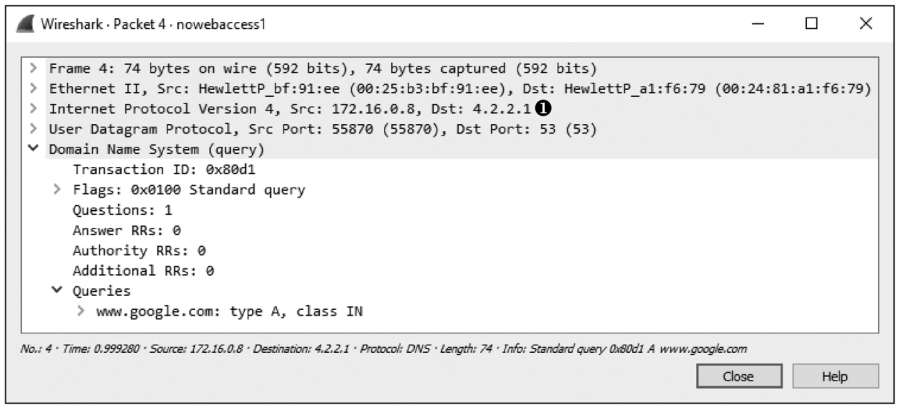
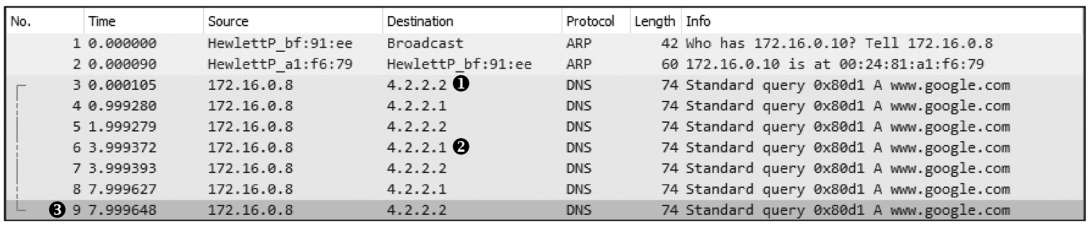


图 10-15　发送到 4.2.2.1 的第二个 DNS 查询

由于计算机仍然没有从 DNS 服务器收到响应，因此 1s 后，查询被再次发送到 4.2.2.2。如图 10-16 所示，这个过程不断重复，在接下来的几秒钟，计算机交替向配置好的首选 DNS 服务器❶和备用 DNS 服务器❷发送请求。整个过程大概花了 8s ❸，这正是用户的 Internet 浏览器报告该页无法访问之前所花的时间。

异步社区Kensuke(18319066421) 专享 请尊重版权



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	HewlettP_bf:91:ee	Broadcast	ARP	42	Who has 172.16.0.10? Tell 172.16.0.8
2	0.000090	HewlettP_a1:f6:79	HewlettP_bf:91:ee	ARP	60	172.16.0.10 is at 00:24:81:a1:f6:79
3	0.000105	172.16.0.8	4.2.2.2 ❶	DNS	74	Standard query 0x80d1 A www.google.com
4	0.999280	172.16.0.8	4.2.2.1	DNS	74	Standard query 0x80d1 A www.google.com
5	1.999279	172.16.0.8	4.2.2.2	DNS	74	Standard query 0x80d1 A www.google.com
6	3.999372	172.16.0.8	4.2.2.1 ❷	DNS	74	Standard query 0x80d1 A www.google.com
7	3.999393	172.16.0.8	4.2.2.2	DNS	74	Standard query 0x80d1 A www.google.com
8	7.999627	172.16.0.8	4.2.2.1	DNS	74	Standard query 0x80d1 A www.google.com
❸ 9	7.999648	172.16.0.8	4.2.2.2	DNS	74	Standard query 0x80d1 A www.google.com

图 10-16　直到通信结束，重复的 DNS 查询才停止

基于前面看到的数据包，我们可以开始查明问题的根源了。首先，我们看见一个 ARP 请求成功地抵达网络上我们认为的默认网关，由此我们知道网关设备在线并且能连接。我们也知道用户的计算机确实在网络上传输数据包，所以我们可以假设本机的协议栈没有问题。显然当进行 DNS 请求时，问题就发生了。

就这个网络来说，DNS 请求是由 Internet 上的外部服务器（4.2.2.2 或 4.2.2.1）解析的。这意味着，要使解析顺利进行的话，负责将数据包路由到 Internet 的路由器必须成功将 DNS 查询转发到服务器，而且服务器必须响应。否则，就无法用 HTTP 请求 Web 页面。

我们知道其他用户上网都没有问题，这说明网络路由器和远程 DNS 服务器也许不是问题的原因所在。剩下唯一可能的问题来源是用户自己的计算机。

进一步检查这台故障计算机后，我们发现它不接受 DHCP 分配的地址，而是手动配置了地址信息，并且默认网关地址设置错了。被设置为默认网关的地址并不是一台路由器，它不能将 DNS 查询数据包转发到网络之外。

3. 学到的知识

在这个情景中，问题出自一台配置错误的客户端。虽然这个问题相当简单，但它却严重影响了用户。对于缺少网络知识或者不能像我们这样可以快速分析数据包的人而言，排除这么简单的配置错误将花费相当多的时间。你可以看到，数据包分析并不局限于大型和复杂的问题。

注意，由于我们不知道网络上默认网关的 IP 地址，因此 Wireshark 不能准确地识别问题，但它可以告诉我们去哪里找，从而节省了宝贵时间。如果我们先与 ISP 联系或者尝试用其他手段排除远程 DNS 服务器的因素，而不是检查网关路由器，那么我们就能将注意力集中到计算机本身、实际也就是问题的原因上。

异步社区Kensuke(18319066421) 专享 请尊重版权

注意 如果我们能更熟悉这个特定网络的 IP 地址分配方案，就可以更快地分析出结果。如果我们注意到 ARP 请求被发送到与网关路由器不同的 IP 地址上，就能立刻知道问题所在。网络出现问题经常是由这些简单的配置错误造成的，通过分析少量的数据包，通常都能快速解决。

10.3.2 意外重定向

在这个情景中，我们又遇到一位不能在工作站上网的用户。然而，不像之前那个用户，她可以访问 Internet，只是不能访问 Google 主页。每次她想访问 Google 的网站时，都被重定向到一个浏览器页面“该页无法显示”。这个问题只影响她一个人。

与之前的情景一样，这是一个只有一些简单交换机和一个简单路由器网关的小型网络。

1. 侦听线路

我们一边监听流量，一边让用户尝试浏览 Google 主页，得到 nowebaccess2.pcap 文件。

2. 分析

如图 10-17 所示，捕获记录文件以一个 ARP 请求和响应开始。在数据包 1 中，用户计算机的 MAC 地址是 00:25:b3:bf:91:ee，IP 地址是 172.16.0.8，它向网段上的所有计算机发送一个 ARP 广播数据包，试图获得主机 172.16.0.102 的 MAC 地址。我们目前还不认识这个地址。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	00:25:b3:bf:91:ee	ff:ff:ff:ff:ff:ff	ARP	42	Who has 172.16.0.102? Tell 172.16.0.8
2	0.000334	00:21:70:c0:56:f0	00:25:b3:bf:91:ee	ARP	60	172.16.0.102 is at 00:21:70:c0:56:f0

图 10-17 对网络上另一个设备的 ARP 请求和响应

在数据包 2 中，用户的计算机了解到 IP 地址 172.16.0.102 的 MAC 地址是 00:21:70:c0:56:f0。根据之前的情形，我们猜测这是网关路由器的地址，通过这个地址数据包可以被再次转发到外部 DNS 服务器。然而，如图 10-18 所示，下一个数据包并不是 DNS 请求，而是从 172.16.0.8 到 172.16.0.102 的 TCP 数据包。它设置了 SYN 标志，表明这是两台主机间建立 TCP 连接时握手的第一个数据包❶。

异步社区Kensuke(18319066421) 专享 请尊重版权

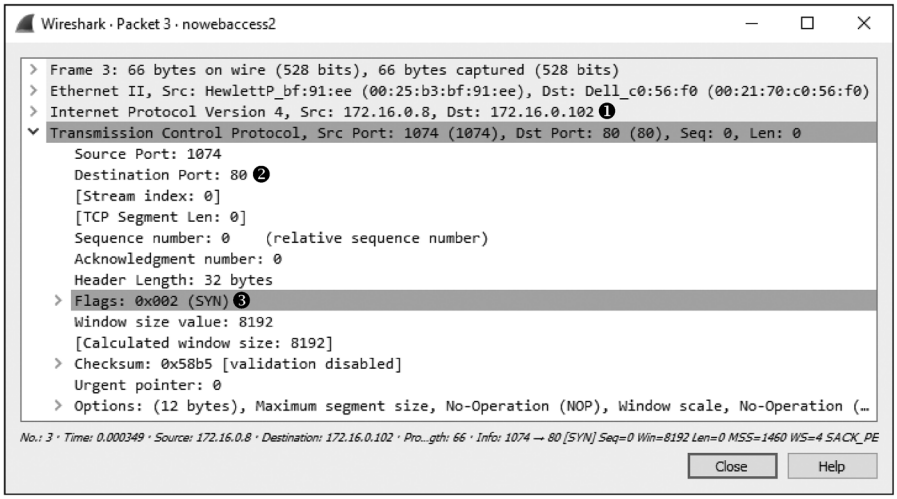


图 10-18　从一台内网主机发往另一台内网主机的 TCP SYN 数据包

显然，试图连接到 172.16.0.102❸的 80 端口❷的 TCP 连接通常与 HTTP 流量有关。如图 10-19 所示，当主机 172.16.0.102 发送回带有 RST 和 ACK 标志❶的 TCP 数据包（数据包 4）时，连接请求就中断了。

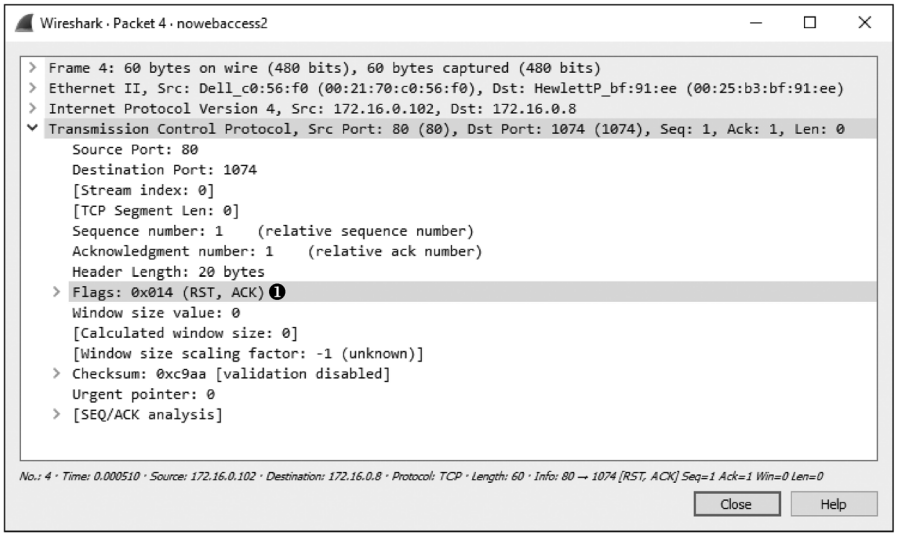
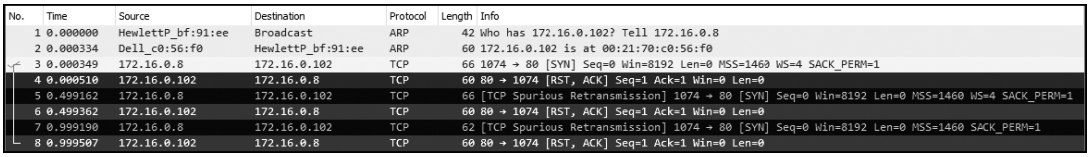


图 10-19　响应 TCP SYN 的 TCP RST 数据包

第 6 章介绍过，带有 RST 标志的数据包是用来结束 TCP 连接的。在这个场景中，主机 172.16.0.8 尝试与主机 172.16.0.102 的 80 端口建立 TCP 连接。不幸

异步社区Kensuke(18319066421) 专享 请尊重版权

的是，由于那台主机没有配置好服务在 80 端口的监听请求，因此只能发送 TCP RST 数据包结束连接。这个过程又重复了两次。如图 10-20 所示，在通信最终结束前，用户计算机发送了一个 SYN 数据包并得到 RST 响应。



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	HewlettP_bf:91:ee	Broadcast	ARP	42	Who has 172.16.0.102? Tell 172.16.0.8
2	0.000334	Dell_c0:56:f0	HewlettP_bf:91:ee	ARP	60	172.16.0.102 is at 00:21:70:c0:56:f0
3	0.000349	172.16.0.8	172.16.0.102	TCP	66	1074 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
4	0.000510	172.16.0.102	172.16.0.8	TCP	60	80 → 1074 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	0.499162	172.16.0.8	172.16.0.102	TCP	66	[TCP Spurious Retransmission] 1074 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
6	0.499362	172.16.0.102	172.16.0.8	TCP	60	80 → 1074 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7	0.999190	172.16.0.8	172.16.0.102	TCP	62	[TCP Spurious Retransmission] 1074 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
8	0.999507	172.16.0.102	172.16.0.8	TCP	60	80 → 1074 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

图 10-20　TCP SYN 和 RST 数据包一共出现了 3 次

此时，用户在浏览器上看到了“该页无法显示”。

在查看其他工作正常的网络设备的配置信息后，数据包 1 和 2 中的 ARP 请求和响应引起了我们的注意。因为 ARP 请求并不是指向网关路由器的真实 MAC 地址，而是其他未知设备。在 ARP 请求和响应之后，我们期望看到向 DNS 服务器的请求，以得到 Google 的 IP 地址，但最终并没有看到。阻止 DNS 查询的两个条件如下。

- 发起连接的设备在 DNS 缓存中已经有域名—IP 地址的对应项。
- 发起连接的设备在 hosts 文件中已经有域名—IP 地址的对应项。

进一步检查这台计算机后，我们发现它的 hosts 文件有一个 Google 表项，对应一个内网 IP 地址 172.16.0.102。这个错误表项就是用户问题的根源。

计算机通常都把 hosts 文件当作域名—IP 地址配对的可信来源，并且会在查询外部来源之前检索它。在这个场景中，用户计算机检查它的 hosts 文件，发现有一个 Google 的表项，就认为 Google 在这个本地网段。接着，它向这个主机发送 ARP 请求，并得到响应，然后尝试向 172.16.0.102 的 80 端口发起 TCP 连接。然而，由于该系统并没有配置成 Web 服务器，因此它不可能接受这个连接请求。

将这个 hosts 文件的表项移除后，用户的计算机就能正常访问 Google 了。

注意　在 Windows 系统上查看 hosts 文件，请打开 C:\Windows\System32\drivers\hosts。在 Linux 上则应查看/etc/hosts。

实际上，这个场景非常普遍。恶意软件在几年前就使用这个方法，把用户

异步社区Kensuke(18319066421) 专享 请尊重版权

重定向到存放恶意代码的网站。试想，如果黑客修改了你的 hosts 文件，每次你登录网上银行，实际上访问的却是一个伪造的网站，专门偷你账户里的钱，这该有多恐怖！

3. 学到的知识

继续分析流量，你会了解各种各样的协议如何工作以及如何阻断它们。在这个场景中，主机没有发送 DNS 请求是因为客户端被错误配置了，而不是因为其他外部限制或外部的错误配置。

在数据包层面上考查这个问题，我们可以迅速发现未知的 IP 地址，也能迅速发现 DNS 这个通信过程的关键部分消失了。通过这些信息，我们可以指出客户端才是问题的来源。

10.3.3 上游问题

与前两个场景一样，在这个场景中，有一位用户抱怨它的工作站无法上网。后来，他发现只是无法访问 Google 这个网站。进一步调查之后，我们发现这个问题影响到了机构的每一个人——谁也无法访问 Google。

这个网络的配置和前两个场景一样，仍然是用一些简单交换机和一个路由器将网络连接到 Internet。

1. 侦听线路

为了解决这个问题，我们首先访问 Google 以生成流量。这是一个全网问题——意味着它也影响你的计算机，而且可能是感染恶意软件导致的——所以你不应该直接在你的设备上嗅探。当你在现实中遇到类似这样的问题时，网络分流器就是较好的解决方案，因为它允许你在短暂中断服务后完全被动地获取流量。通过网络分流器获得的流量被保存在 nowebaccess3.pcap 文件中。

2. 分析

这个数据包的捕获以 DNS 流量开始，而不是我们之前看到的 ARP 流量。因为捕获的第一个数据包发往一个外部地址，并且数据包 2 包含来自那个地址的响应，所以我们可以假设 ARP 过程已经完成了，并且网关路由器的 MAC-IP 地址映射已经存在于主机的 ARP 缓存中。

如图 10-21 所示，捕获中的第一个数据包从主机 172.16.0.8 发往地址 4.2.2.1❶，并且它是一个 DNS 数据包❷。查看该数据包的内容，我们发现这是一个查询

异步社区Kensuke(18319066421) 专享 请尊重版权

Google 的 A 记录请求❸。

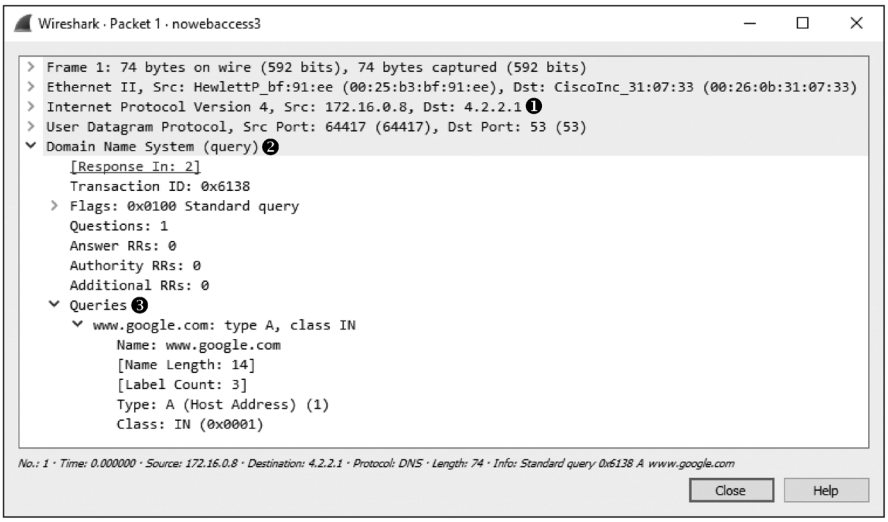


图 10-21　查询 Google 的 A 记录

如图 10-22 所示，来自 4.2.2.1 的响应是捕获文件的第 2 个数据包。查看 Packet Details 面板，我们发现响应这个请求的域名服务器提供了多个回答❶。此时看起来通信一切正常。

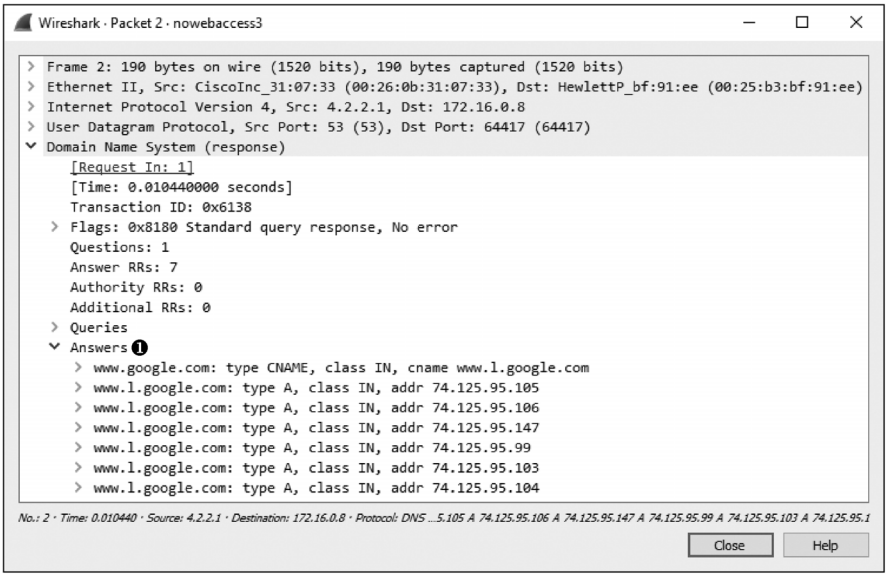


图 10-22　包含多个 A 记录的 DNS 响应

异步社区Kensuke(18319066421) 专享 请尊重版权

现在用户的计算机已经得到 Web 服务器的 IP 地址，它可以尝试与服务器通信了。如图 10-23 所示，通信过程从数据包 3 开始，这是一个从 172.16.0.8 发往 74.125.95.105 的 TCP 数据包❶。这个目标地址来自数据包 2 中 DNS 查询响应提供的第 1 个 A 记录。TCP 数据包设置了 SYN 标志❷，并尝试连接远程服务器的 80 端口❸。

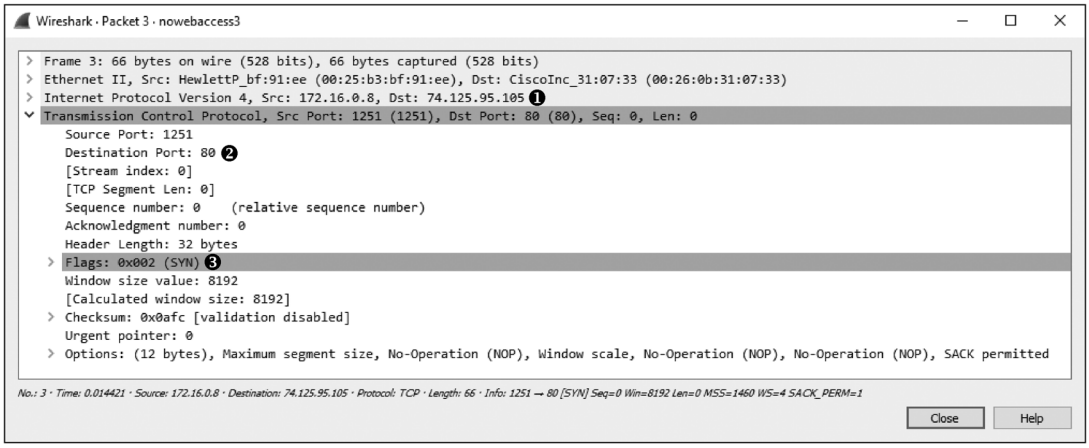


图 10-23　尝试连接 80 端口的 SYN 数据包

因为这是一个 TCP 握手过程，所以我们知道应该在响应中看到 TCP SYN/ACK 数据包，但是主机过一会儿又发送了另一个 SYN 数据包到目标。这个过程大概在 1s 后再次发生，如图 10-24 所示，到这里通信停止了，浏览器报告找不到网站。

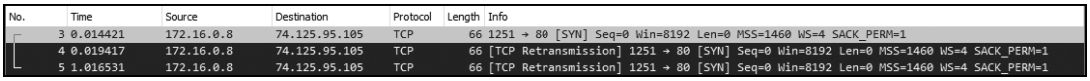


图 10-24　TCP SYN 数据包尝试了 3 次都没有收到响应

这时，我们想到由于能成功向外部 DNS 服务器提交查询请求，因此网络内的工作站可以连接到外网。DNS 服务器响应了一些看起来有效的地址，然后我们的主机就尝试向其中一个地址建立连接。而且，我们尝试连接的本地工作站看起来功能正常。

问题是远程服务器没有响应我们的连接请求，连 TCP RST 数据包都没发过来。可能的原因有几种：Web 服务器配置错误、Web 服务器的协议栈崩溃、远程网络部署了数据包过滤设备[1]。假设本地网络没有数据包过滤设备，那么所有

[1] 比如防火墙。——译者注

异步社区Kensuke(18319066421) 专享 请尊重版权

可能的解决方法都在远程网络上，这超出了我们的控制范围。在这个案例中，Web 服务器不能正常工作，我们的所有尝试都失败了。一旦 Google 修复故障[1]，通信就可以继续了。

3. 学到的知识

这个场景中的问题不是我们能修复的。我们的分析表明，问题不在于我们网络上的主机、路由器，也不在于提供域名解析服务的外部 DNS 服务器。问题在我们的网络设施之外。

有时候发现这不是我们的问题不仅能缓解压力，也能在管理层来敲门时挽回颜面。我与很多运营商、设备厂商和软件公司打过交道，他们都说不是自己那边的问题，但你已经看到，数据包是不会说谎的。

10.4 打印机故障

我们的技术支持管理员遇到了一个打印机的难题。销售部门的用户报告说大批量的打印机出故障了。当用户发送大量打印作业给它时，它会打印几页然后停止工作。他们改动了很多驱动配置选项，但没有任何效果。技术支持的员工们希望你去确认这不是网络问题。

10.4.1 侦听线路

这个问题的焦点在于打印机，因此我们希望尽可能将嗅探器放到离打印机最近的地方。虽然我们不能在打印机上安装 Wireshark，但网络上使用了高级三层交换机，所以我们可以使用端口镜像功能。我们将打印机连接的端口镜像到一个空端口，然后将一台装有 Wireshark 的笔记本电脑连接到该端口。安装完成后，我们让一位用户发送大量打印作业给打印机，而我们会监视端口输出内容，最后得到捕获记录文件 inconsistent_printer.pcap。

10.4.2 分析

如图 10-25 所示，捕获文件的开头是发送打印作业的主机（172.16.0.8）与打印机（172.16.0.253）的 TCP 握手。握手之后，一个大小为 1460 字节的 TCP 数据包发送到打印机❶。数据大小既可以在 Packet List 面板 Info 列的右边看到，

[1] 某些情况下并非 Google 的故障，而是访问路径中存在某种防火墙。——译者注

异步社区Kensuke(18319066421) 专享 请尊重版权

也可以在 Packet Details 面板的 TCP 头部信息的底部看到。

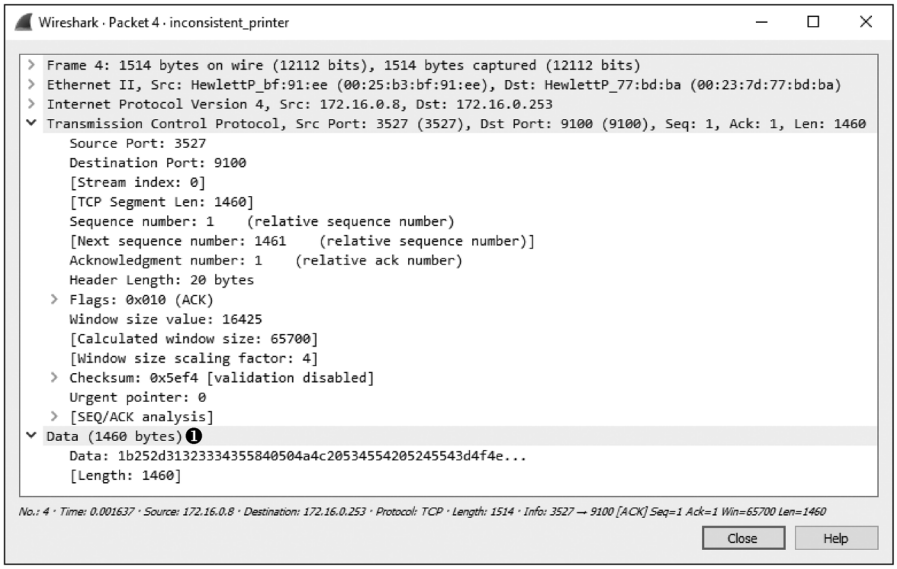


图 10-25　通过 TCP 传输到打印机的数据

数据包 4 后面是另一个包含 1460 字节的数据包❶，如图 10-26 所示。这个数据被打印机❷确认了。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.16.0.8	172.16.0.253	TCP	66	3527 → 9100 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
2	0.000166	172.16.0.253	172.16.0.8	TCP	66	9100 → 3527 [SYN, ACK] Seq=0 Ack=1 Win=8760 Len=0 MSS=1460 WS=1 SACK_PERM=1
3	0.000201	172.16.0.8	172.16.0.253	TCP	54	3527 → 9100 [ACK] Seq=1 Ack=1 Win=65700 Len=0
4	0.001637	172.16.0.8	172.16.0.253	TCP	1514	3527 → 9100 [ACK] Seq=1 Ack=1 Win=65700 Len=1460
5	0.001646	172.16.0.8	172.16.0.253	TCP	1514	3527 → 9100 [ACK] Seq=1461 Ack=1 Win=65700 Len=1460 ❶
❷ 6	0.005493	172.16.0.253	172.16.0.8	TCP	160	9100 → 3527 [PSH, ACK] Seq=1 Ack=2921 Win=7888 Len=106
7	0.005561	172.16.0.8	172.16.0.253	TCP	1514	3527 → 9100 [ACK] Seq=2921 Ack=107 Win=65592 Len=1460
8	0.005571	172.16.0.8	172.16.0.253	TCP	1514	3527 → 9100 [ACK] Seq=4381 Ack=107 Win=65592 Len=1460
9	0.005578	172.16.0.8	172.16.0.253	TCP	1514	3527 → 9100 [ACK] Seq=5841 Ack=107 Win=65592 Len=1460
10	0.005585	172.16.0.8	172.16.0.253	TCP	1514	3527 → 9100 [ACK] Seq=7301 Ack=107 Win=65592 Len=1460
11	0.033569	172.16.0.253	172.16.0.8	TCP	60	9100 → 3527 [ACK] Seq=107 Ack=8761 Win=6144 Len=0
12	0.033626	172.16.0.8	172.16.0.253	TCP	1514	3527 → 9100 [ACK] Seq=8761 Ack=107 Win=65592 Len=1460
13	0.033640	172.16.0.8	172.16.0.253	TCP	1514	3527 → 9100 [ACK] Seq=10221 Ack=107 Win=65592 Len=1460
14	0.033649	172.16.0.8	172.16.0.253	TCP	1514	3527 → 9100 [ACK] Seq=11681 Ack=107 Win=65592 Len=1460
15	0.033658	172.16.0.8	172.16.0.253	TCP	1514	3527 → 9100 [ACK] Seq=13141 Ack=107 Win=65592 Len=1460
16	0.098314	172.16.0.253	172.16.0.8	TCP	60	9100 → 3527 [ACK] Seq=107 Ack=14601 Win=4400 Len=0

图 10-26　正常的数据传输和 TCP 确认

在捕获文件的最后两个数据包之前，数据流一直正常。数据包 121 是一个 TCP 重传数据包，也是故障的第一个标志，如图 10-27 所示。

当一个设备发送 TCP 数据包给远程设备，而远程设备没有确认此次传输时，就发送一个 TCP 重传数据包。一旦到达重传门限，发送设备就假设远程设备没有收到数据，从而立刻重传数据包。在通信停止之前，这个过程重复了多次。

异步社区Kensuke(18319066421) 专享 请尊重版权

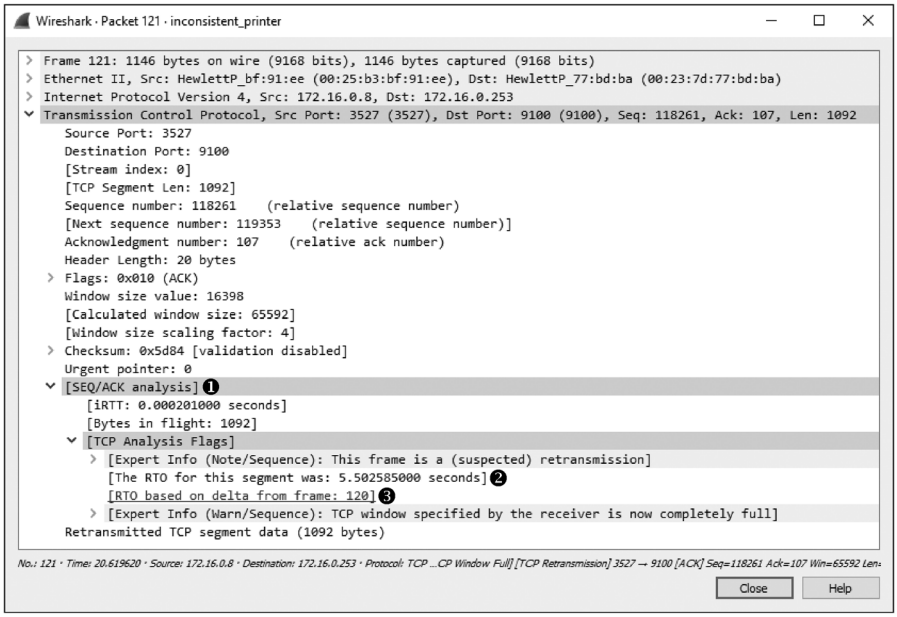


图 10-27　这些 TCP 重传数据包是故障的一个标志

在这个场景中，因为打印机没有确认传输的数据，所以客户工作站就向打印机发送重传数据包。如果展开 TCP 头部的 SEQ/ACK analysis 部分以及下方的额外信息，如图 10-27 所示❶，你就可以从细节中看到为什么这是重传。根据 Wireshark 加工的细节，数据包 121 是数据包 120 的重传❸。另外，重传数据包的重传超时（RTO）在 5.5s ❷左右。

当分析数据包间隔时间时，你可以更改时间显示格式以适应特定情形。在这个案例中，我们想看一看之前的数据包在发送多久后发生了重传，于是选择 View->Time Display Format 并改成 Seconds Since Previous Captured Packet 这个选项。然后，如图 10-28 所示，你可以清楚地看见初始数据包（数据包 120）发送 5.5s 后发生了数据包 121 的重传❶。

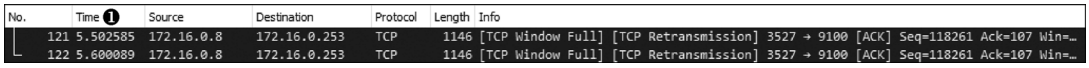


图 10-28　查看数据包间隔时间有利于解决问题

下一个数据包是数据包 120 的另一个重传。这个数据包的 RTO 是 11.10s，包括上一个数据包的 5.5s RTO。Packet List 面板的 Time 列告诉我们，在上一次

异步社区Kensuke(18319066421) 专享 请尊重版权

重传 5.6s 后发生了这次重传。这好像是捕获文件中的最后一个数据包，巧合的是，打印机大概在这个时间停止打印了。

好在这个分析场景只涉及内网的两台设备，所以我们只需要确定是客户工作站还是打印机的问题。我们可以看见数据正常流动了相当长的时间，然而在某一时刻，打印机停止响应工作站了。工作站尽了最大努力投递数据包，重传就是一个明证，但打印机就是没有响应。这个问题可以在其他工作站上重现，所以我们猜测打印机才是问题的来源。

进一步分析后，我们发现打印机的内存出故障了。当大量打印作业发送到打印机时，它只打印一定的页数，一旦访问到特定内存区域就停止工作。由此可见，内存问题导致打印机无法接收新数据，并中断了与主机的通信。

10.4.3 学到的知识

虽然这个打印机问题不是网络所致，但我们仍可以用 Wireshark 指出它。跟之前的场景不同，这里只有 TCP 流量。幸好，当两台设备停止通信时，TCP 给我们留下了有用的信息。

在这个案例中，当通信意外停止时，我们靠 TCP 内置的重传功能指出了问题的确切位置。继续学习下面的场景时，我们会经常依赖于这样的功能来解决更复杂的问题。

10.5 分公司之困

在这个场景中，一家公司在总部之外新开设了几家分公司。通过部署一台 Windows 域控制器服务器和一台备用域控制器，公司的所有 IT 设施几乎都放置在总部。域控制器负责处理 DNS 和分公司用户的认证请求。

域控制器是一个代理 DNS 服务器，它接收来自总部的上游 DNS 服务器的资源记录信息。

当部署团队将新设施延伸到分公司时，发现没有人能访问网络上的内部 Web 应用服务器。这些服务器位于总部办公室，通过广域网（Wide Area Network，WAN）访问。问题影响到分公司的所有用户，并只限于这些内部服务器。所有用户都可以访问 Internet 以及分公司内的其他资源。

图 10-29 显示了在这个场景中要考虑的组件，包括了多个站点。

异步社区Kensuke(18319066421) 专享 请尊重版权

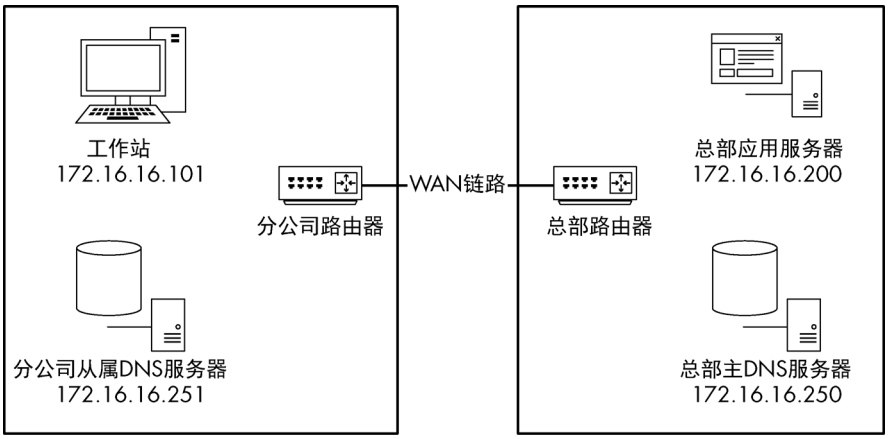


图 10-29　分公司之困问题的相关组件

10.5.1　侦听线路

由于问题出在总部和分公司间的通信过程中，因此我们可以在多个地点收集数据来跟踪问题。问题可能出在分公司的客户端上，所以我们使用端口镜像功能查看其中一台计算机在线路上看到了什么数据包。收集完这个信息后，我们可以使用它推测出其他收集地点以帮助解决问题。从其中一个客户端捕获的初始数据包保存在 stranded_clientside.pcap 文件中。

10.5.2　分析

如图 10-30 所示，当工作站 172.16.16.101 尝试访问托管在总部应用服务器 172.16.16.200 的应用程序时，产生了捕获文件的第 1 个数据包。这个捕获只有两个数据包。第 1 个数据包是发送到 172.16.16.251❶的 DNS 请求，查询应用服务器❸的 A 记录❷。这是总部 172.16.16.200 服务器的 DNS 域名。

如图 10-31 所示，这个数据包的响应是服务器故障❶，表明 DNS 查询被阻止了。注意到这个数据包只是一个错误（服务器故障），并没有响应查询结果❷。

现在我们知道该通信故障与 DNS 有关。因为分公司的 DNS 查询由 DNS 服务器 172.16.16.251 解析，我们前往下一站。

为了从分公司的 DNS 服务器捕获合适的流量，我们将嗅探器留在原地，只改变端口镜像设置。现在服务器的流量就被镜像到我们的嗅探器了。捕获结果在 stranded_branchdns.pcap 文件中。

异步社区Kensuke(18319066421) 专享 请尊重版权

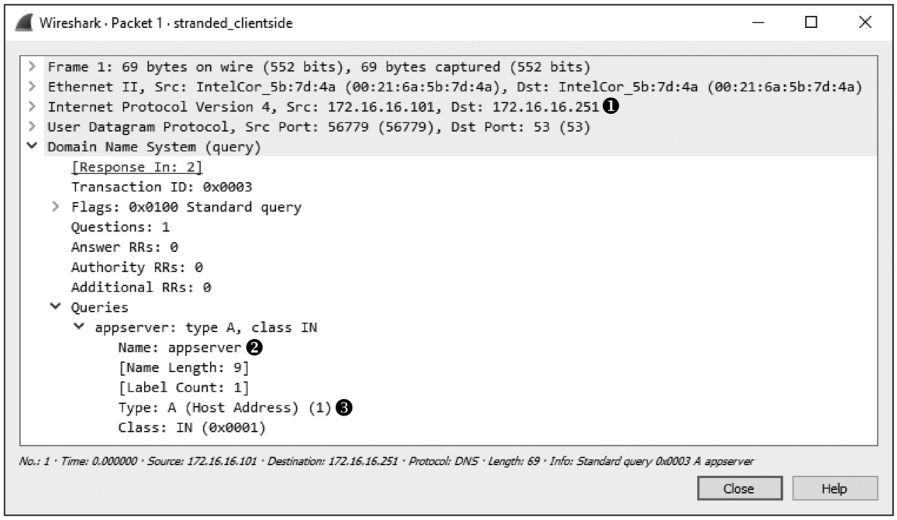


图 10-30　通信从查询应用服务器 A 记录的 DNS 请求开始

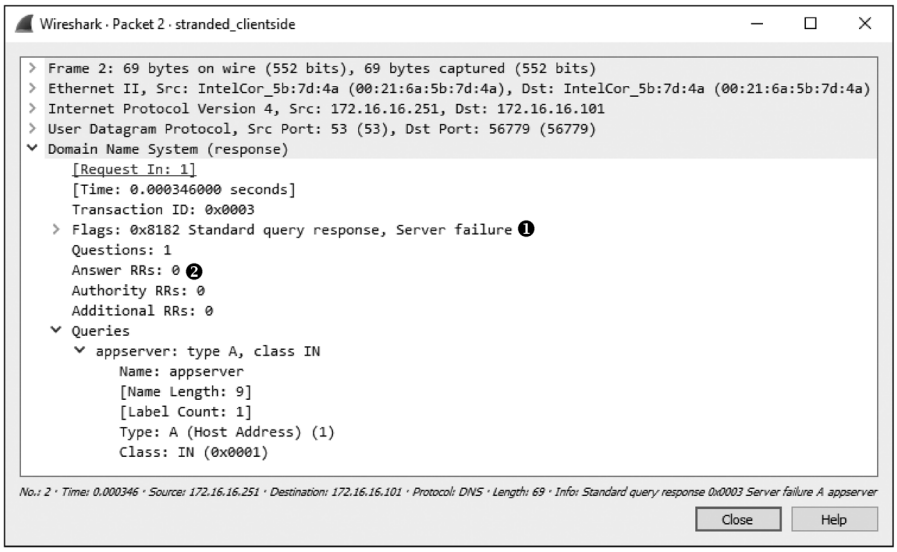


图 10-31　查询响应表明这是上游的问题

如图 10-32 所示，这个捕获的开头是我们之前看到的查询和响应，但还有一个额外的数据包。额外的数据包看起来很奇怪，因为它尝试与中心办公室的首选 DNS 服务器（172.16.16.250）❶的标准 DNS 服务端口 53❷进行通信，但它却不是我们过去看见的 UDP 类型❸。

异步社区Kensuke(18319066421) 专享 请尊重版权

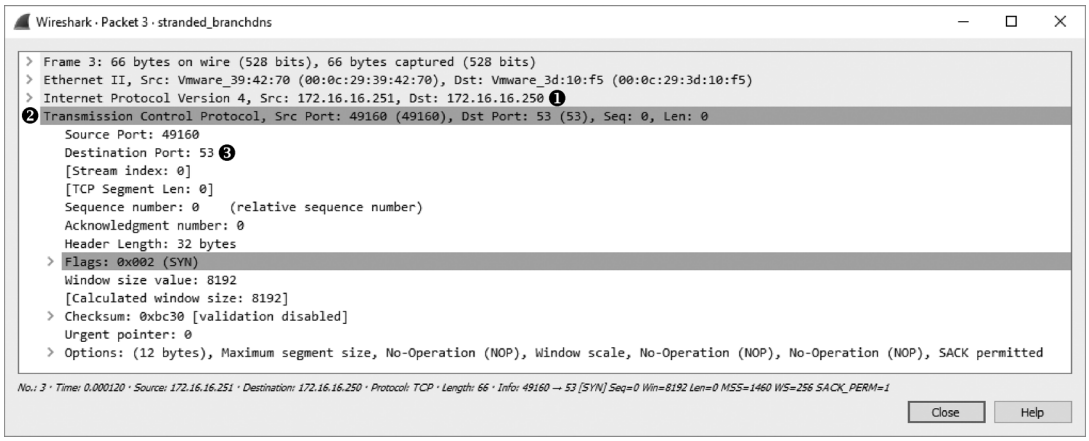


图 10-32　这个 SYN 数据包使用了 53 端口，但不是 UDP

为了找出这个数据包的用途，回顾我们在第 7 章对 DNS 的讨论。DNS 通常使用 UDP，但当响应超过一定大小时要使用 TCP。在那种情况下，我们会看见一些触发 TCP 流量的 UDP 流量。另外，TCP 也用于 DNS 的区域传送过程，它使资源记录在 DNS 服务器之间传输，这里就是该种情况。

分公司的 DNS 服务器是总部 DNS 服务器的从属服务器，意味着分公司的 DNS 服务器依赖于从总部服务器获得资源记录。分公司用户试图访问的应用服务器放置在总部，意味着总部 DNS 服务器是它的权威 DNS 服务器。要使分公司服务器能解析用户对应用服务器的 DNS 请求，总部 DNS 服务器必须把 DNS 资源记录传输给分公司 DNS 服务器，这可能是捕获文件中 SYN 数据包的来源。

SYN 数据包没有得到响应，这告诉我们总部和分公司 DNS 服务器之间失败的区域传送导致了 DNS 故障。现在我们可以进一步找出区域传送失败的原因。办公室之间的路由器或中心办公室的 DNS 服务器可能是罪魁祸首。为了找出问题，我们可以嗅探中心办公室 DNS 服务器的流量，查看 SYN 数据包是不是到达了服务器。

我没有给出中心办公室 DNS 服务器的流量捕获文件，因为根本就没有。SYN 数据包从来没有到达服务器。派遣技术人员查看连接两个办公室的路由器配置后，我们发现中心办公室的路由器被配置成只允许 53 端口的 UDP 流量进入，而 53 端口的 TCP 流量则被阻止了。这个简单的配置错误阻止了服务器间的区域传送，从而导致分支办公室的客户端无法解析对中心办公室设备的查询。

异步社区Kensuke(18319066421) 专享 请尊重版权

10.5.3 学到的知识

看完这个“犯罪剧”，你一定学到了很多关于调查网络通信问题的知识。当“犯罪”发生后，侦探开始问讯受害者。找到线索，顺藤摸瓜，直到找到罪魁祸首。

在这个场景中，我们一开始先查看了受害者（工作站），然后找到了 DNS 通信问题这个线索。这个线索将我们带到分支 DNS 服务器，然后又到中心 DNS 服务器，最终找到路由器，也就是问题的来源。

在分析时，请尝试从数据包中找出线索。线索不一定能告诉你谁是“罪犯”，但通常它们最终能帮你找出来。

10.6 生气的开发者

在 IT 界，开发者和系统管理员经常争吵。开发者总是将程序故障归咎于糟糕的网络设置和设备。系统管理员则倾向于把网络错误和网络缓慢归咎于糟糕的代码。

在这个场景中，程序员开发了一个应用程序，用于跟踪多个商店的销售并将报告发回中心数据库。为了节约正常工作时间的带宽，它没有被设计成实时应用程序。而是等报告数据累积一天后，才在晚上以逗号分隔值（comma-separated value，CSV）文件的形式传回，插入中心数据库中。

然而，这个新开发的应用程序工作情况不太正常。服务器接收到了各个商店传回的文件，但插入数据库的数据是错误的。一些地区的数据丢失了，有的数据还存在错误，而且文件的某些部分还丢失了。系统管理员很烦恼，因为程序员抱怨这是网络的问题。程序员一口咬定文件在从商店传到中心数据库时被损坏了。我们就要证明他是错的。

10.6.1 侦听线路

为了收集所需数据，我们可以在其中一个商店或中心办公室捕获数据包。故障影响到了所有商店，因此如果这确实是网络导致的问题，那肯定是在中心办公室那边——它是所有商店通信的汇聚点。

因为网络交换机支持端口镜像，所以我们用端口镜像功能嗅探服务器的

异步社区Kensuke(18319066421) 专享 请尊重版权

流量。我们只捕获单个商店上传 CSV 文件到收集服务器的流量。结果保存在 tickedoffdeveloper.pcap 文件中。

10.6.2 分析

除了网络上的基本信息流量之外，我们完全不了解程序员开发的应用程序。捕获文件看起来是以一些 FTP 流量开始的，因此我们将调查这是不是传输 CSV 文件采用的机制。简洁、干净的通信最适合查看通信流量图了。选择 Statistics -> Flow Graph，然后单击“OK”。图 10-33 显示了结果图像。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.16.16.128 ❶	172.16.16.121 ❷	TCP	66	2555 → 21 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
2	0.000071	172.16.16.121	172.16.16.128	TCP	66	21 → 2555 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
3	0.000242	172.16.16.128	172.16.16.121	TCP	60	2555 → 21 [ACK] Seq=1 Ack=1 Win=17520 Len=0
4	0.002749	172.16.16.121	172.16.16.128	FTP	96	Response: 220 FileZilla Server version 0.9.34 beta
5	0.002948	172.16.16.128	172.16.16.121	FTP	70 ❸	Request: USER salesxfer
6	0.003396	172.16.16.121	172.16.16.128	FTP	91	Response: 331 Password required for salesxfer
7	0.003514	172.16.16.128	172.16.16.121	FTP	69	Request: PASS p@ssw0rd
8	0.004862	172.16.16.121	172.16.16.128	FTP	69	Response: 230 Logged on

图 10-33　流量图提供了 FTP 通信的快速视图

首先观察一下包列表（见图 10-34），我们会看到一个 172.16.16.128❶与 172.16.16.121❷之间的基本 FTP 连接。由于 172.16.16.128 发起连接，因此我们猜测它是客户端，172.16.16.121 则是汇总与处理数据的服务器。流量图确认这些流量只用到了 FTP 协议，在握手竞争之后，我们开始看到来自客户端的 FTP 请求和来自服务器的响应❸。

我们知道这里会有一些数据传输，因此我们用 FTP 的知识，定位开始传输数据的位置。FTP 连接和数据传输是由客户端发起的，因此我们应该寻找用于上传数据到 FTP 服务器的 FTP STOR 命令。简单的方法是生成一个过滤器。

这个捕获文件里到处都是 FTP 请求命令，因此我们不需要面对表达式生成器中的数百个协议和选项，只要在 Packet List 面板中直接生成过滤器就行了。为此，我们首先需要选择一个出现有 FTP 请求命令的数据包。我们选择了数据包 5，这是最接近列表顶部的一个。然后展开 Packet Details 面板的 FTP section 和 USER section。右击 Request Command：USER 域，并选择 Prepare a Filter。最后，选择 Selected。

这将生成一个筛选含有 FTP USER 请求命令的数据包的过滤器，并出现在过滤器对话框中。接着，如图 10-34 所示，编辑过滤器，将单词 USER 替换成 STOR ❶。

现在按回车键应用这个过滤器，你会看见捕获文件里只有一个 STOR 命令，

异步社区Kensuke(18319066421) 专享 请尊重版权

在数据包 64 ❷中。

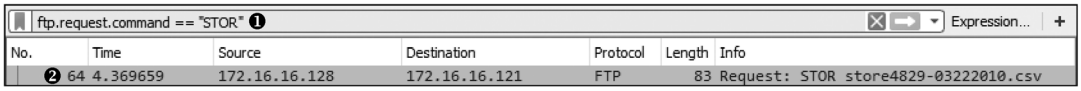


图 10-34　这个过滤器有助于识别数据从哪里开始传输

既然我们已经知道数据从哪里开始传输了，就可以单击 Packet List 面板上方的 Clear 按钮清除过滤器。

查看从数据包 64 开始的捕获文件，我们看见这个数据包指定传输 store4829-03222010.csv 文件❶，如图 10-35 所示。

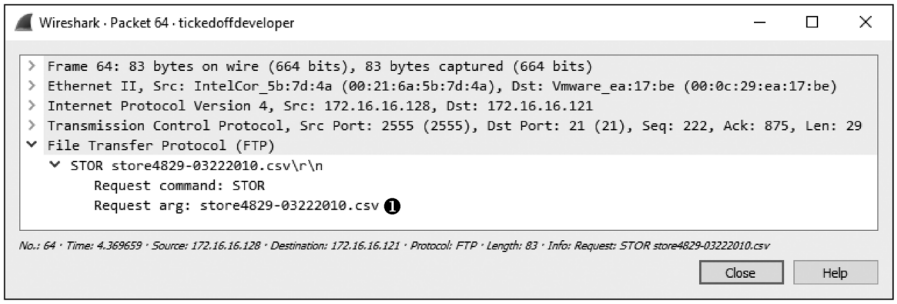


图 10-35　使用 FTP 传输 CSV 文件

STOR 命令后面的数据包使用了不同的端口，但它们被识别成 FTP 数据传输的一部分。我们已经验证数据在传输了，但我们仍然没有证明程序员是错的。为此，我们需要从捕获的数据包中提取传输文件，以展示文件在网络中传输后并没有被损坏。

当文件以未加密格式在网络中传输时，它会被分解成多个段，并在目的地被重新组装。在这个场景中，我们在数据包到达目的地并且尚未被组装之时捕获它们。数据就在那儿了，我们只需要将文件提取为数据流来重新组装它。为此，选择 FTP 数据流的任一个数据包（比如数据包 66），并单击 Follow TCP Stream。结果显示在 TCP 流中，如图 10-36 所示。

由于数据在 FTP 中以明文传输，所以我们能看到它，但却不能仅由此断定文件是完整的。为了重组数据，以便将其提取为原始格式，我们单击 Save As 按钮并指定数据包 64 显示的文件名，如图 10-37 所示。然后单击 Save。

保存操作的结果应该是一个 CSV 文件，这是对商店系统传过来的文件的字节层次的复制。我们通过比较原始文件和提取文件的 MD5 哈希值来验证该文

异步社区Kensuke(18319066421) 专享 请尊重版权

件。MD5 哈希值应该是一样的，如图 10-37 所示。

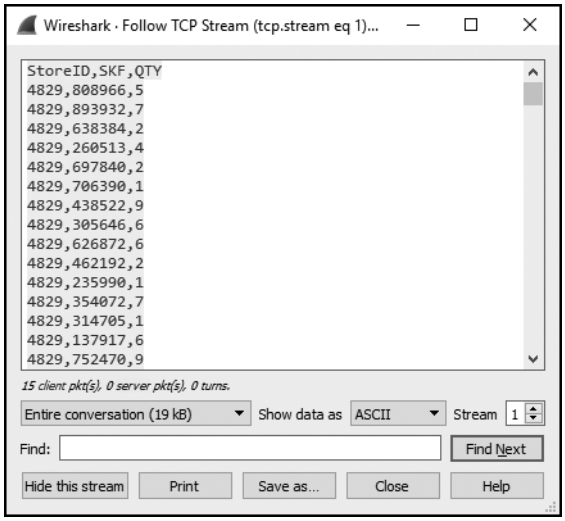


图 10-36　TCP 流显示传输的数据将数据流保存为原始文件名

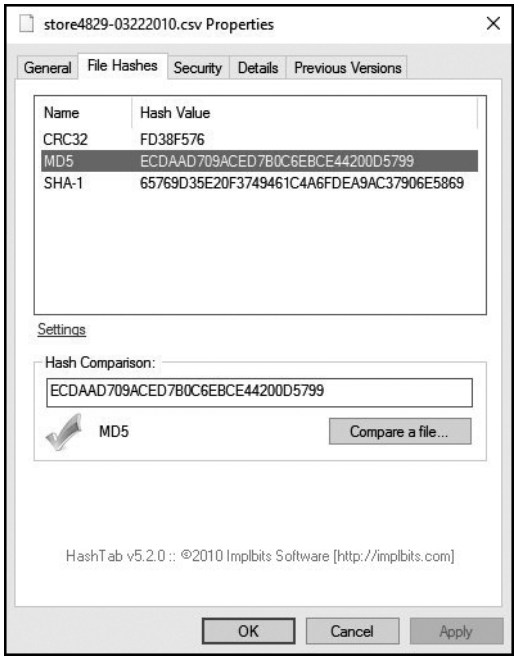


图 10-37　原始文件和提取文件的 MD5 哈希值相等

通过比较文件，我们可以证明网络并不是应用程序数据库出错的原因。文件

异步社区Kensuke(18319066421) 专享 请尊重版权

从商店传输到收集服务器时是完整的，所以肯定是应用程序处理文件时出错了。

10.6.3 学到的知识

数据包层面分析的一个好处是你不必处理杂乱无章的应用程序。糟糕的程序远比好的程序要多，但在数据包层面，它们就无所谓了。在这个案例中，程序员关心的是应用程序所依赖的所有不明组件，但最终，他那耗费数百行代码写成的数据传输不过就是 FTP、TCP 和 IP 而已。通过使用我们了解的基本协议知识，可以确认通信过程毫无差错，甚至能通过提取文件证明网络正常。记住这个关键的结论：无论手里的问题多么复杂，都只是一些数据包而已。

10.7 结语

在本章，我们讲述了几个基本场景，数据包分析使我们更好地理解了通信故障。通过分析常见协议，我们可以及时查出并解决网络问题。你在现实网络中可能不会遇到跟这里完全相同的场景，但本章讨论的分析技术应该能给你一些有益启发。

异步社区Kensuke(18319066421) 专享 请尊重版权

第11章

让网络不再卡



作为一名网络管理员，你将花费很多时间用于修复运行缓慢的计算机和服务。但是人们抱怨网络缓慢，并不意味着就是网络的问题。

在开始处理网络缓慢的问题之前，你首先要确定网络是否真的很慢。你将在本章中学到这些技巧。

首先，我们会讨论 TCP 的错误恢复和流量控制机制。然后，我们会探索如何检测网络缓慢的根源。最后，我们会讨论用基线测试网络以及网络上运行的设备与服务的方法。读完本章后，你在识别、诊断和解决慢速网络方面，应该会有非常大的进步。

注意

很多技术都可以用来排除网络缓慢故障。本章内容主要集中在 TCP，因为在大多数时间你只需要面对它。TCP 允许你执行被动地回溯分析，而不用生成额外的流量（比如 ICMP）。

异步社区Kensuke(18319066421) 专享 请尊重版权

11.1 TCP 的错误恢复特性

TCP 的错误恢复特性是我们定位、诊断并最终修复网络高延迟的最好工具。在计算机网络中，“延迟”是数据包传输与接收时间差的衡量参数。

延迟可以被测量为单程延迟（从单个来源到一个目的地）或往返延迟（从来源到达目的地并返回来源）。当设备间通信很快，并且数据包从一个端点到另一端点所花时间很少时，就说通信是低时延的。相反，当数据包在来源和目的地间传输要花费大量时间时，就说通信是高时延的。高时延是所有珍视自己声誉（以及工作）的网络管理员们的头号敌人。

在第 6 章中，我们讨论了 TCP 如何使用序号和确认号来保证可靠地传递数据包。在本章，我们将再次关注序号和确认号，观察当高时延导致这些号码乱序抵达（或根本没有接收到）时，TCP 是如何响应的。

11.1.1 TCP 重传

重传数据包是 TCP 最基本的错误恢复特性之一，它被设计用来对付数据包丢失。

数据包丢失可能有很多原因，包括出故障的应用程序、流量负载沉重的路由器或者临时性的服务中断。数据包层次上的移动速度非常快，而且数据包丢失通常是暂时的，因此 TCP 能否检测到数据包丢失并从中恢复显得至关重要。

决定是否有必要重传数据包的主要机制叫作重传计时器。这个计时器负责维护一个叫重传超时（Retransmission timeout，RTO）的值。每当使用 TCP 传输一个数据包时，就启动重传计时器。当收到这个数据包的 ACK 时，计时器就会停止。从发送数据包到接收 ACK 确认之间的时间被称为往返时间（Round-trip time，RTT）。将若干个这样的时间平均下来，可算出最终的 RTO 值。

在最终算出 RTO 值之前，传输操作系统将一直依赖于默认配置的 RTT 值。此项设定用于主机间的初始通信，并基于接收到的数据包 RTT 进行调整，以形成真正的 RTO。

一旦 RTO 值确定下来，重传定时器就被用于每个传输的数据包，以确定数据包是否丢失。图 11-1 阐述了 TCP 重传过程。

异步社区Kensuke(18319066421) 专享 请尊重版权

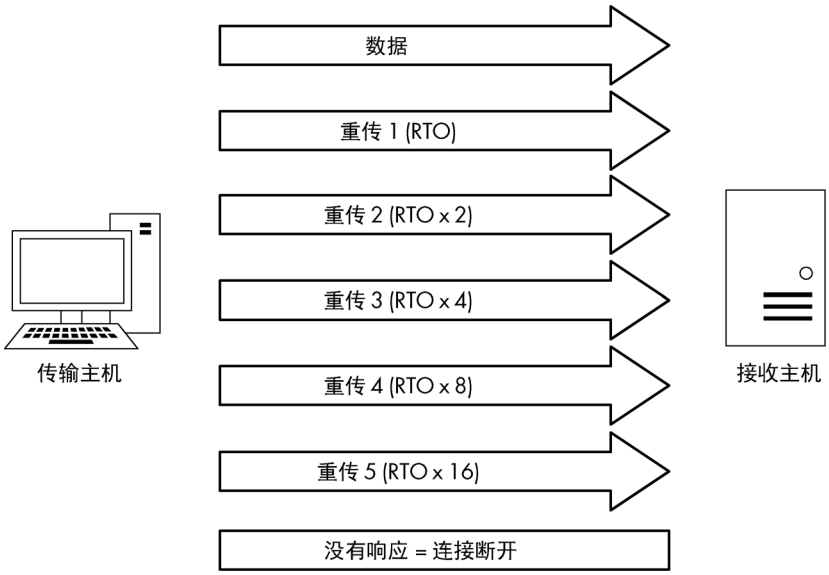


图 11-1　TCP 重传过程的概念视图

当数据包被发送出去，但接收方没有发送 TCP ACK 数据包时，传输主机就假设原来的数据包丢失了，并重传它。重传之后，RTO 值翻倍。如果在到达那个值之前一直没有接收到 ACK 数据包，则将发生另一次重传。如果下一次重传还是没有收到 ACK，那么 RTO 值将翻倍。每次重传，RTO 值都将翻倍，这个过程会持续到收到一个 ACK 数据包，或者发送方达到配置的最大重传次数为止。

最大重传次数取决于传输操作系统上的配置。默认情况下，Windows 主机最多重传 5 次，大部分 Linux 主机则默认重传 15 次。这个选项在两个操作系统中都是可配置的。

要看 TCP 重传的例子，请打开 tcp_retransmissions.pcap 文件，它包含了 6 个数据包。第一个数据包如图 11-2 所示。

这是一个 TCP PSH/ACK 数据包❶，包含 648 字节的数据❷，从 10.3.30.1 发送到 10.3.71.7❸。这是一个典型的数据包。

在正常条件下，你会期待在发送第一个数据包之后，很快就能看到响应的 TCP ACK 数据包。然而，在这个例子中，下一个数据包是一次重传。通过在 Packet List 面板中查看这个数据包，你就能得出这个结论。Info 列明确表明了[TCP Retransmission]，并且这个数据包以黑底红字出现。图 11-3 显示了 Packet List 面板中列出的重传例子。

异步社区Kensuke(18319066421) 专享 请尊重版权

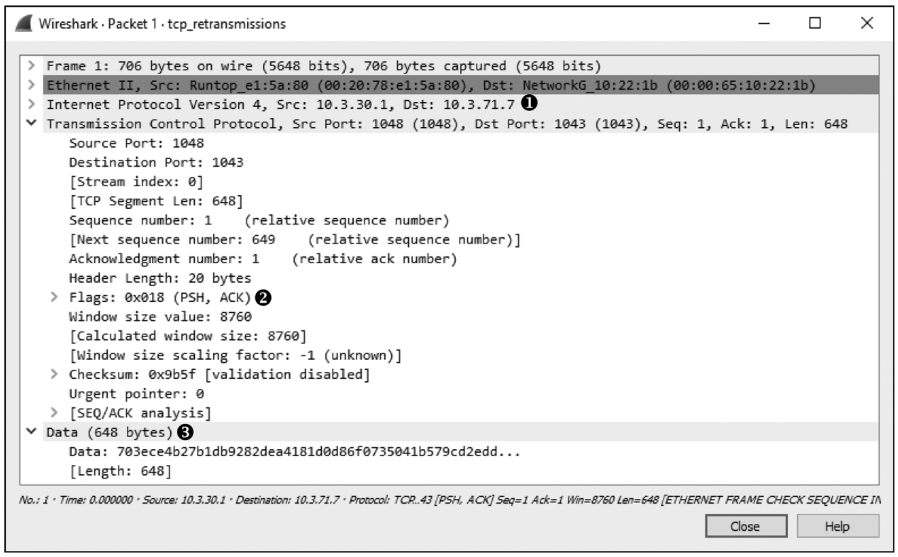


图 11-2　包含数据的简单 TCP 数据包

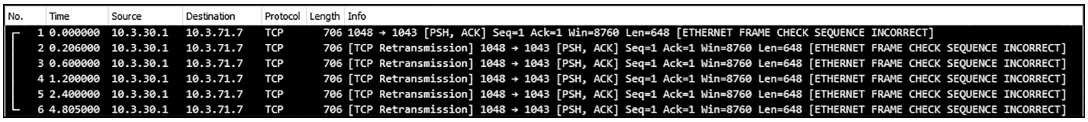


图 11-3　Packet List 面板中的重传

如图 11-4 所示，你也可以通过查看 Packet Details 和 Packet Bytes 面板确定它是否是重传数据包。

注意，除了 IP identification 和 Checksum 域之外，这个数据包与最初的数据包完全一致。为了验证这个结论，可在 Packet Bytes 面板中比较这个重传数据包和最初的数据包❶。

在 Packet Details 面板中，注意到重传数据包的 SEQ/ACK Analysis 标题下有一些额外的信息❷。这个有用的信息是由 Wireshark 提供的，实际上并不包含在数据包里。SEQ/ACK analysis 告诉我们这确实是一个重传❸，RTO 值 0.206s ❹是基于与数据包 1❺的时间差值算出来的。

查看剩下的数据包应该是类似的结果，唯一的不同在于 IP identification、Checksum 域以及 RTO 值。为了显示每个数据包之间的时间间隔，如图 11-5 所示，可以查看 Packet List 面板的 Time 列。在这里，你看到每一次重传后 RTO 值翻倍，时间呈指数增长。

异步社区Kensuke(18319066421) 专享 请尊重版权

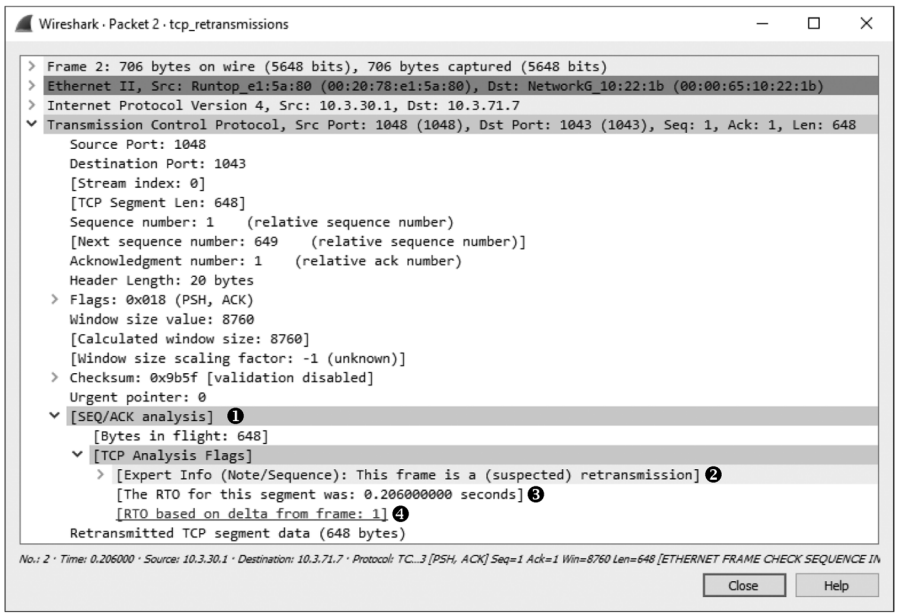


图 11-4　一个重传数据包

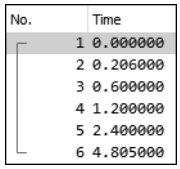


图 11-5　Time 列显示了 RTO 值的增长

传输设备使用 TCP 的重传特性来检测数据包丢失并从中恢复。下一步，我们将查看 TCP 的重复确认特性，它被接收方用于检测数据包丢失并从中恢复。

11.1.2　TCP 重复确认和快速重传

当接收方收到乱序数据包时，就发送重复的 TCP ACK 数据包。TCP 在其头部使用序号和确认号域，以确保数据被可靠接收并以发送顺序重组。

注意　“TCP 数据包”的准确术语其实应该是“TCP 区段”，但大多数人倾向于把它们称为“数据包”。

异步社区Kensuke(18319066421) 专享 请尊重版权

建立一个新的 TCP 连接时，初始序号（Initial sequence number，ISN）是握手过程中交换的最重要信息之一。一旦设置好连接两端的 ISN，接下来传输的每一个数据包都将按照数据载荷的大小增长序号。

举个例子，一台主机的 ISN 是 5000，它发送一个 500 字节的数据包给接收方。一旦接收到此数据包，接收方就会根据以下规则响应一个包含确认号 5500 的 TCP ACK 数据包：

接收数据的序号+接收数据的字节数=发出的确认号

在这个运算中，返回到发送方的确认号实际上就是接收方期待下次接收的数据包序号。图 11-6 中可以看到一个这样的例子。

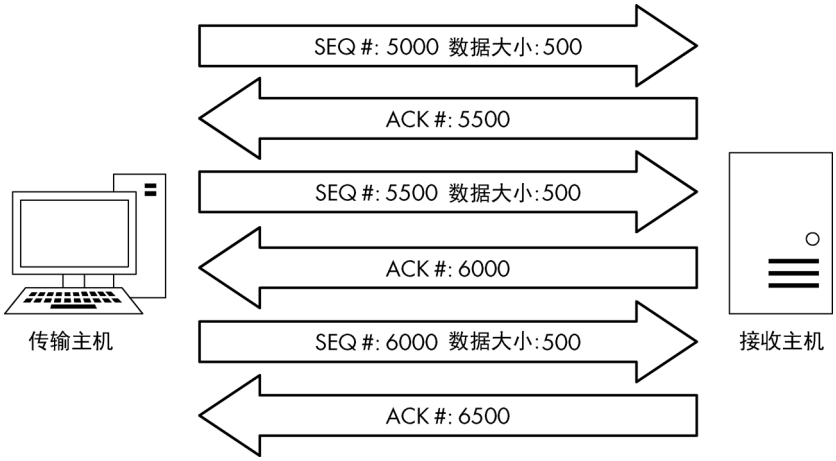


图 11-6　TCP 序号和确认号

序列号使数据接收方检测数据包丢失成为可能。当接收方追踪正在接收的序号时，如遇到不合顺序的序号，它就知道数据包丢失了。

当接收方收到一个预料之外的序号时，它会假设有一个数据包在传输中丢失了。为了正确重组数据，接收方必须要得到丢失的数据包，因此它重新发送一个包含丢失数据包的序号的 ACK 数据包，以通知发送方重传该数据包。

当传输主机收到 3 个来自接收方的重复 ACK 时，它就假设这个数据包确实在传输中丢失了，并立刻发送一个快速重传。一旦触发快速重传，其他所有正在传输的数据包都要靠边，直到把快速重传数据包发送出去为止。图 11-7 描述了这个过程。

异步社区Kensuke(18319066421) 专享 请尊重版权

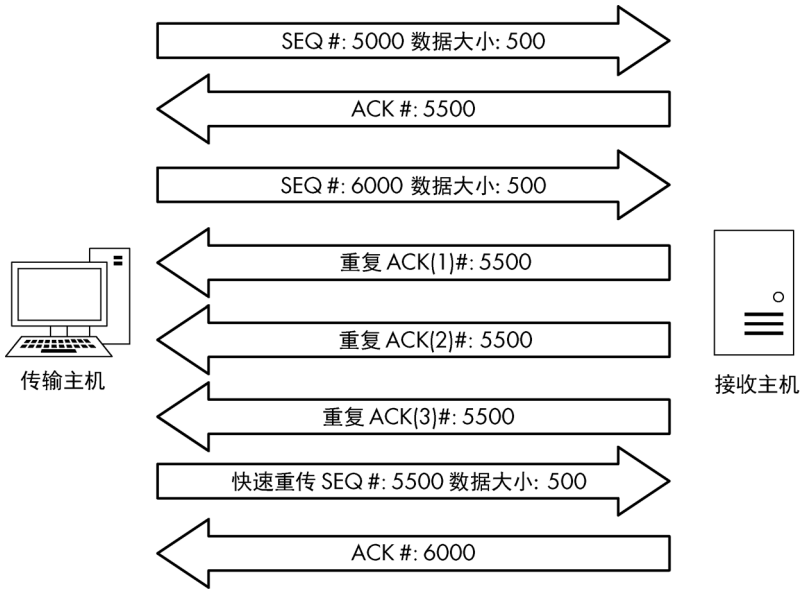


图 11-7　来自接收方的重复 ACK 导致快速重传

你将在 tcp_dupack.pcap 文件中发现重复 ACK 和快速重传的例子。捕获记录中的第一个数据包，如图 11-8 所示。

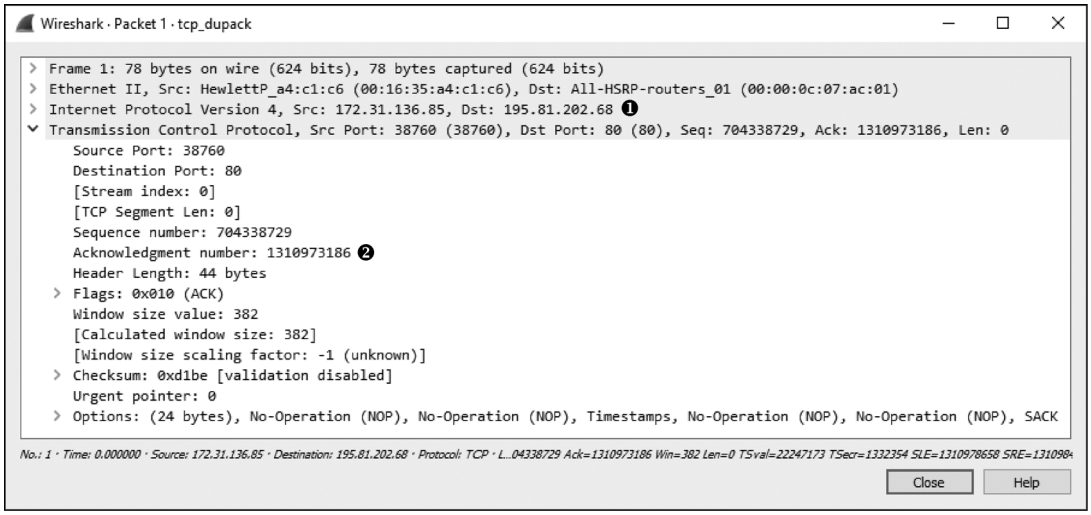


图 11-8　ACK 显示了下一个期待的序号

在网络中这个 TCP ACK 数据包从数据接收方（172.31.136.85）去往发送方

异步社区Kensuke(18319066421) 专享 请尊重版权

（195.81.202.68）❶，包含了一个对捕获文件之前的数据包的确认。

注意　默认情况下，Wireshark 使用相对序号来简化对这些数字的分析，但在接下来的几节中，并未在例子和截图中使用这个特性。使用如下方法可关闭此项功能，选择 Edit->Preferences，在 Preferences 窗口中选择 Protocols，然后选择 TCP 区段，最后取消 Relative sequence numbers 和 window scaling 旁边的复选框。

如图 11-9 所示，此数据包中的确认号是 1310973186❷，这应该是接收的下一个数据包的序号。

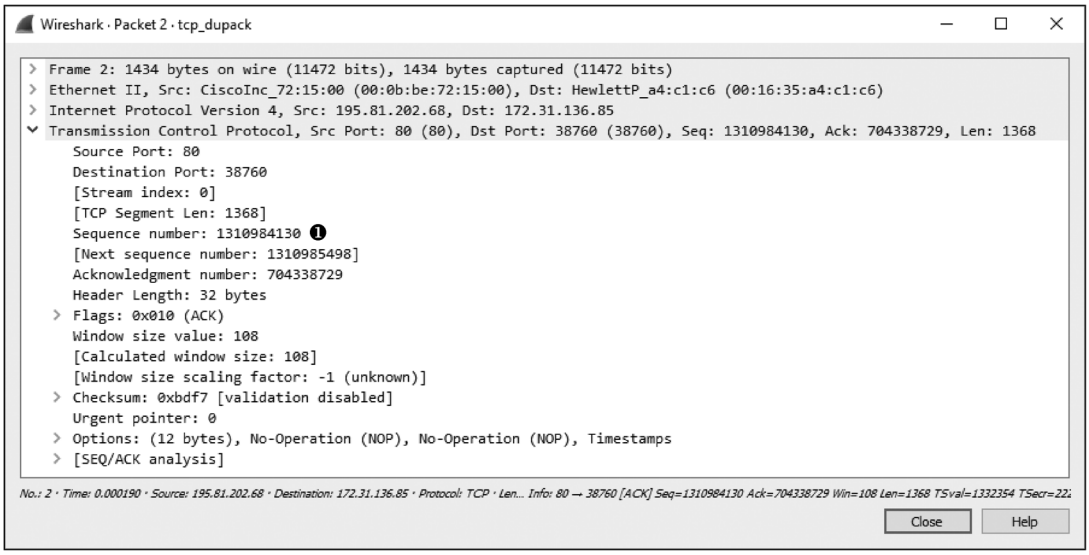


图 11-9　此数据包的序号与预料的不同

很遗憾，下一个数据包的序号是 1310984130 ❶，并非是我们所期待的。这表明期待的数据包在传输中莫名其妙地丢失了。如图 11-10 所示，接收主机注意到这个数据包的序号不符，就在捕获记录的第三个数据包中发送一个重复的 ACK。

通过查看以下信息的其中一个，你就可以确定这是一个重复的 ACK 数据包。

- Packet Details 面板中的 Info 列。这个数据包以黑底红字呈现。
- SEQ/ACK Analysis heading 下的 Packet Details 面板。若展开此标题，你会发现这个数据包被列为数据包 1 的重复 ACK。

异步社区Kensuke(18319066421) 专享 请尊重版权

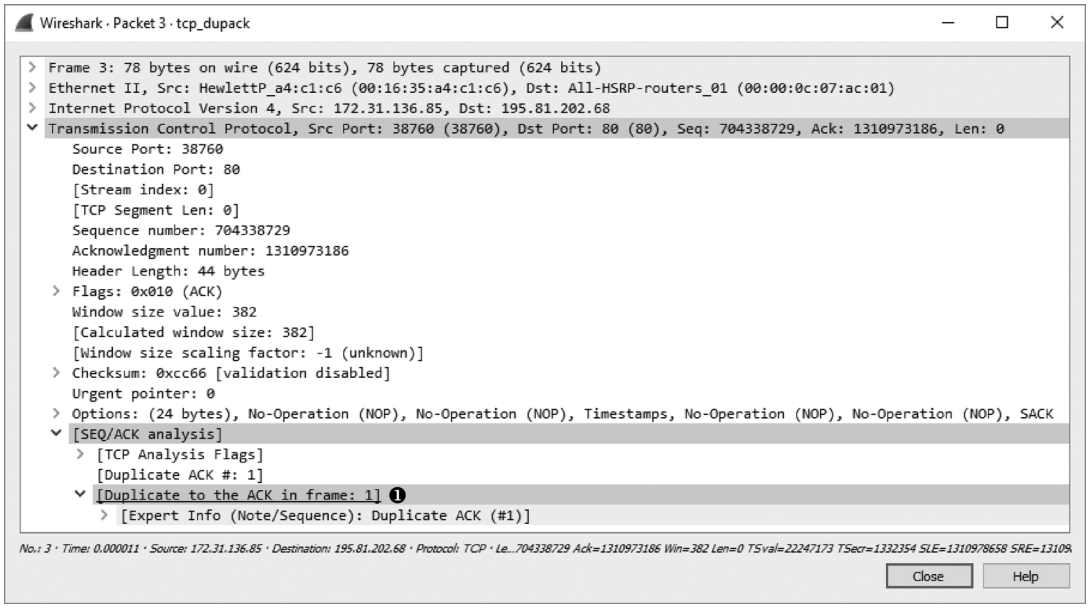


图 11-10　第一个重复 ACK 数据包

如图 11-11 所示，接下来的几个数据包继续这个过程。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.31.136.85	195.81.202.68	TCP	78	38760 → 80 [ACK] Seq=704338729 Ack=1310973186 Win=382 Len=0 TSval=22247173 TSecr=1332354 …
2	0.000190	195.81.202.68	172.31.136.85	TCP	1434	80 → 38760 [ACK] Seq=1310984130 Ack=704338729 Win=108 Len=1368 TSval=1332354 TSecr=222471…
3	0.000011	172.31.136.85	195.81.202.68	TCP	78	[TCP Dup ACK 1#1] 38760 → 80 [ACK] Seq=704338729 Ack=1310973186 Win=382 Len=0 TSval=22247…
❶ 4	0.000093	195.81.202.68	172.31.136.85	TCP	1434	80 → 38760 [ACK] Seq=1310985498 Ack=704338729 Win=108 Len=1368 TSval=1332354 TSecr=222471…
❷ 5	0.000010	172.31.136.85	195.81.202.68	TCP	78	[TCP Dup ACK 1#2] 38760 → 80 [ACK] Seq=704338729 Ack=1310973186 Win=382 Len=0 TSval=22247…
❸ 6	0.000121	195.81.202.68	172.31.136.85	TCP	1434	80 → 38760 [ACK] Seq=1310986866 Ack=704338729 Win=108 Len=1368 TSval=1332354 TSecr=222471…
❹ 7	0.000010	172.31.136.85	195.81.202.68	TCP	78	[TCP Dup ACK 1#3] 38760 → 80 [ACK] Seq=704338729 Ack=1310973186 Win=382 Len=0 TSval=22247…

图 11-11　由于乱序数据包的影响，生成了额外的重复 ACK

捕获文件的第 4 个数据包是发送主机以错误序号发送的另一个数据区块❶。因此，接收主机发送第二个重复 ACK❷。接收方又收到一个包含错误序号的数据包❸。这导致它传输第三个、也是最后一个重复 ACK❹。

发送方收到来自接收方的第三个重复 ACK 之后，就强制停止所有的数据包传输，并重新发送丢失的数据包。图 11-12 显示了丢失数据包的快速重传。

在 Packet List 面板的 Info 列中再次出现了重传数据包。正如前面的例子，数据包被清楚地标记为黑底红字。这个数据包的 SEQ/ACK 分析部分告诉我们这有可能是一次快速重传❶（再次注意，数据包的快速重传标记信息并非数据包本身的值，而是 Wireshark 的功能）。捕获记录的最后一个数据包是确认收到快速重传的 ACK 数据包。

异步社区Kensuke(18319066421) 专享 请尊重版权

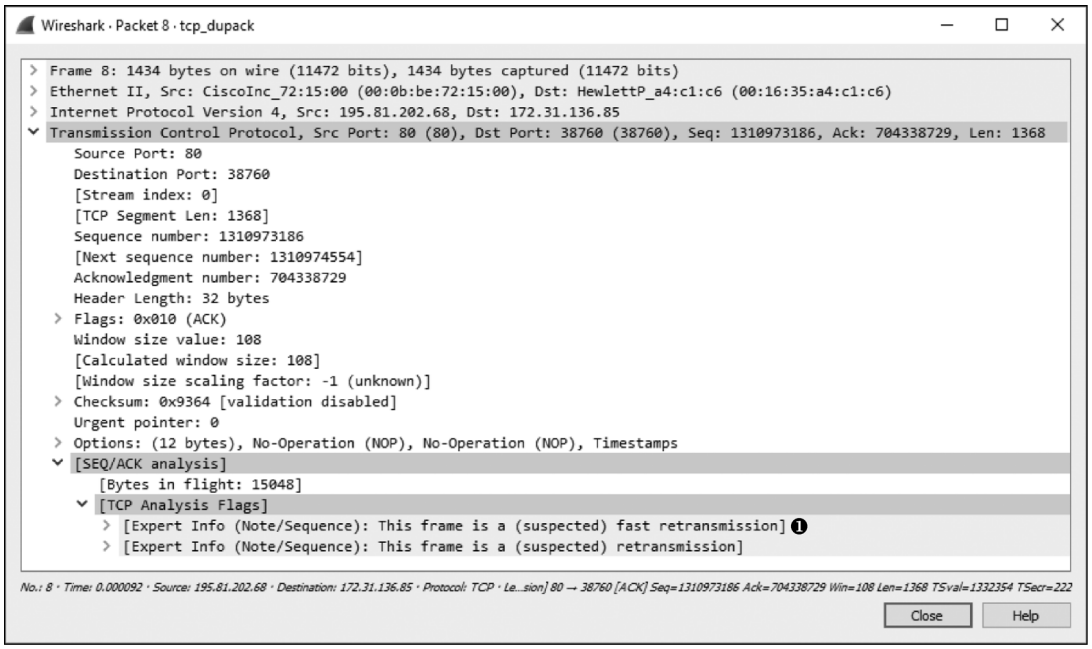


图 11-12　重复 ACK 引发了丢失数据包的快速重传

注意　当发生数据包丢失时，可能影响 TCP 通信数据流的功能是选择性确认（Selective Acknowledgement）。在上面的捕获记录里，通信双方已经在三次握手过程中协商开启了选择性 ACK。因此，一旦数据包丢失并收到重复 ACK，即使在丢失数据包之后还是成功接收了其他数据包，也只需要重传丢失的数据包。如果不启用选择性 ACK，那就必须重新传输丢失数据包之后的每一个数据包。选择性 ACK 使得数据丢失的恢复更加高效。由于大部分现代 TCP/IP 协议栈的实现都支持选择性 ACK，因此你会发现这个功能通常都会被启用。

11.2　TCP 流控制

重传和重复 ACK 都是 TCP 反应性的功能，被设计用来从数据包丢失中恢复。如果 TCP 没有包含某些形式的用于预防数据包丢失的前瞻性功能，那么它将是一个糟糕的协议，但幸好它做到了。

TCP 实现了滑动窗口机制，用于检测何时发生了数据包丢失，并调整数据

异步社区Kensuke(18319066421) 专享 请尊重版权

传输速率加以避免。滑动窗口机制利用数据接收方的接收窗口来控制数据流。

接收窗口是数据接收方指定的值，存储在 TCP 头部（以字节为单位），它告诉发送设备自己希望在 TCP 缓冲空间中存储多少数据。这个缓冲空间是数据在可以向上传递到等待处理数据的应用层协议之前的临时存储空间。因此，发送方一次只能发送窗口大小域指定的数据量。为了传输更多的数据，接收方必须发送确认，以告知之前的数据已经接收到了。它也必须要处理占用 TCP 缓冲空间的数据，以清空缓冲区。图 11-13 阐明了接收窗口是如何工作的。

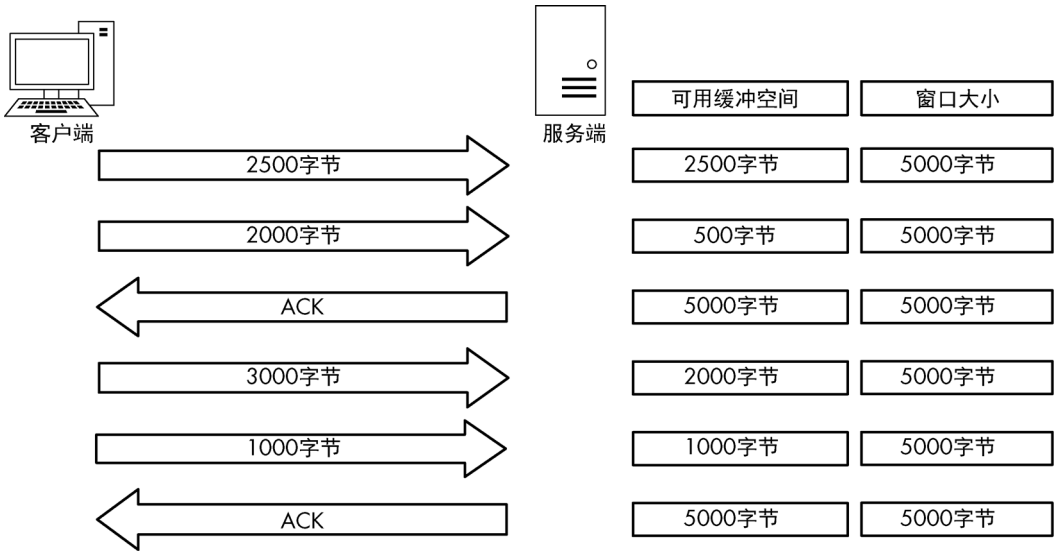


图 11-13　接收窗口使得接收方不被数据淹没

在图 11-13 中，客户端正在向接收窗口大小为 5000 字节的服务器发送数据。客户端发送了 2500 字节，将服务器的可用缓冲空间减少到 2500 字节，然后再发送 2000 字节，进而将可用缓冲区减少到 500 字节。然后服务器送出这些数据的确认，它处理缓冲区的数据，得到可用的空缓冲区。这个过程不断重复，客户端又发送了 3000 字节和另外 1000 字节，将服务器的可用缓冲区减少到 1000 字节。客户端再次确认这些数据，并处理缓冲区的内容。

11.2.1　调整窗口大小

调整窗口大小的过程相当明确，但有时候它也不尽如人意。每当接收到数据时，TCP 栈就生成并发送一个确认作为响应，但是接收方不可能总是迅速地处理缓冲区的数据。

异步社区Kensuke(18319066421) 专享 请尊重版权

当一台繁忙的服务器处理来自多个客户端的数据包时，服务器可能会因缓慢地清空缓冲区，而腾不出空间来接收新数据。如果没有流量控制，这将导致数据包丢失和数据损坏。幸好，当服务器太繁忙，以致不能以宣告的接收窗口的速率处理数据时，它可以调整接收窗口的大小。它通过减小向发送方返回的ACK 数据包的 TCP 头部窗口大小值达到这个目的。图 11-14 展示了这样的例子。

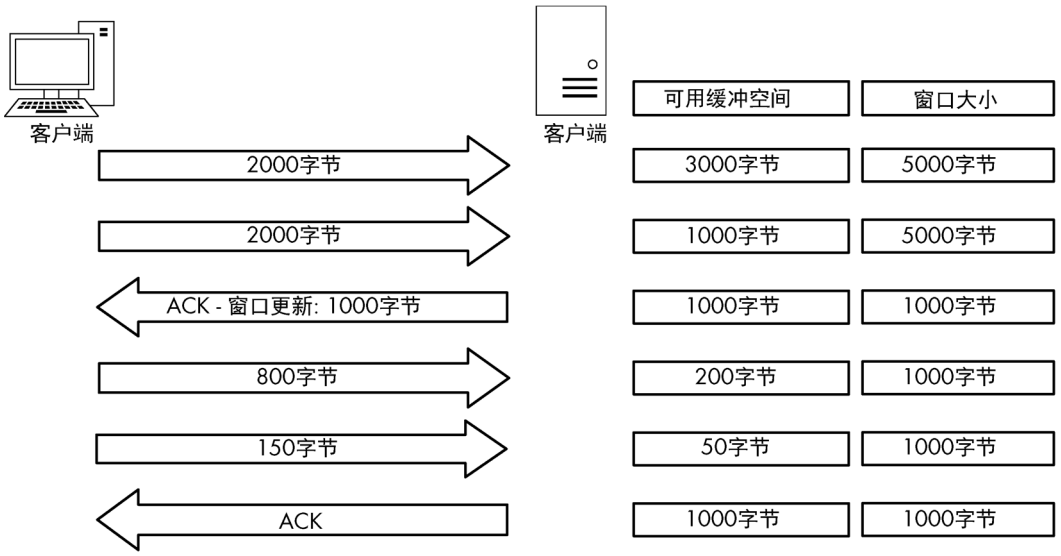


图 11-14　服务器变繁忙时，可以调整窗口大小

在图 11-14 中，服务器一开始声明的窗口大小是 5000 字节。客户端发送了 2000 字节，紧接着再发送 2000 字节，只留下 1000 字节的可用缓冲空间。服务器意识到它的缓冲区很快就要被塞满了。它知道如果按此速率传输数据，那数据包很快就会丢失。为了校正这个问题，服务器向客户端发送一个确认，包含更新的窗口大小为 1000 字节。因而，客户端会发送较少的数据，服务端可以按照能接受的速率处理缓冲区的内容，从而允许数据恒定流动。

窗口重设过程是双向工作的。当服务器能更快地处理数据时，它可以发送一个 ACK 数据包，指明更大的窗口大小。

11.2.2　用零窗口通知停止数据流

某些情况下，服务器可能无法处理客户端发送的数据。这可能是因为内存不足、缺少处理能力或者其他问题。这可能导致数据包中止、通信过程停止，

异步社区Kensuke(18319066421) 专享 请尊重版权

但接收窗口能够将负面影响最小化。

当出现这种情况时，服务器可以发送一个数据包，指明窗口大小是零。当客户端收到这个数据包时，它会停止所有数据传输，但仍通过传输“保活数据包（keep-alive packets）”保持与服务器的连接。客户端周期性地发送保活数据包，以检查服务器接收窗口的状态。一旦服务器能再次处理数据，它就会响应一个非零的窗口大小，这时通信恢复。图 11-15 显示了零窗口通知的例子。

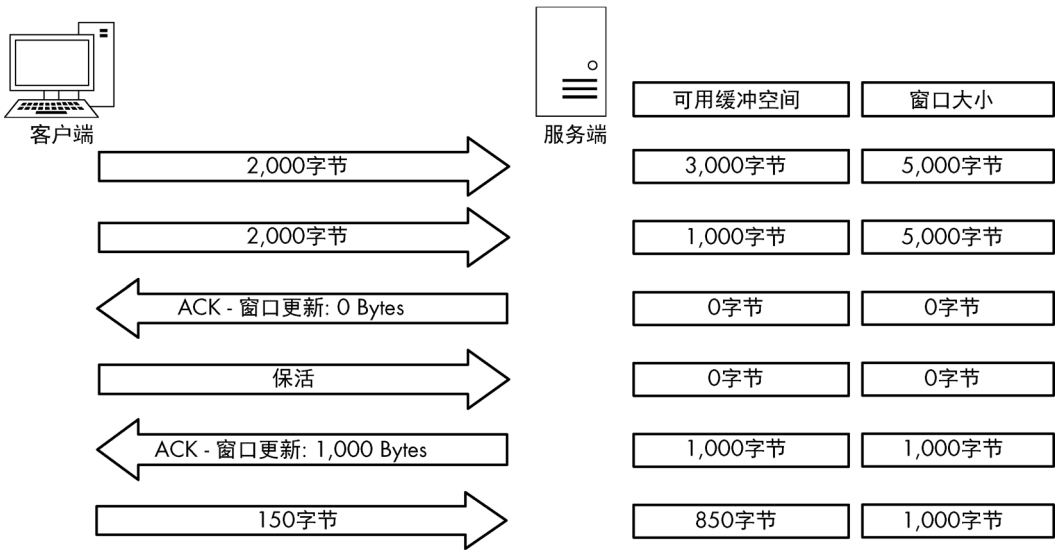


图 11-15　当窗口大小设为 0 字节时，数据传输就停止了

在图 11-15 中，服务器开始用 5000 字节的窗口接收数据。在从客户端接收到 4000 字节的数据后，服务器开始承受沉重的处理负担，不能再处理来自客户端的任何数据。这时服务器发送一个数据包，将窗口大小域设为 0。客户端暂停数据传输，并发送一个保活数据包。收到保活数据包之后，服务器响应一个数据包，通知客户端它现在可以接收数据了，而且它现在的窗口大小是 1000 字节。客户端继续发送数据。

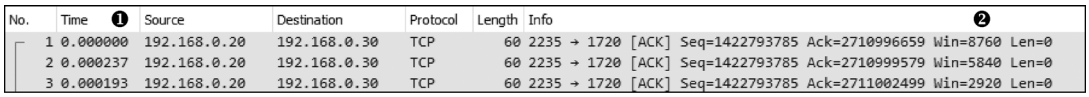
11.2.3　TCP 滑动窗口实战

看完 TCP 滑动窗口的理论之后，我们将在捕获文件 tcp_zerowindowrecovery.pcap 中探究它。

在这个文件中，我们从 192.168.0.20 发送给 192.168.0.30 的几个 TCP ACK

异步社区Kensuke(18319066421) 专享 请尊重版权

数据包开始。我们主要对 Windows Size 域感兴趣，可以在 Packet List 面板的 Info 列以及 Packet Details 面板的 TCP 头部看到它。从图 11-16 可以立即发现，在前面的 3 个数据包中，这个域的值不断减小。



No.	Time ❶	Source	Destination	Protocol	Length	Info ❷
1	0.000000	192.168.0.20	192.168.0.30	TCP	60	2235 → 1720 [ACK] Seq=1422793785 Ack=2710996659 Win=8760 Len=0
2	0.000237	192.168.0.20	192.168.0.30	TCP	60	2235 → 1720 [ACK] Seq=1422793785 Ack=2710999579 Win=5840 Len=0
3	0.000193	192.168.0.20	192.168.0.30	TCP	60	2235 → 1720 [ACK] Seq=1422793785 Ack=2711002499 Win=2920 Len=0

图 11-16　这些数据包的窗口大小在递减

这个值从第一个数据包的 8760 字节减少到第二个数据包的 5840 字节，接着又减为第三个数据包的 2920 字节❶。窗口大小值递减是主机延迟增加的典型指标。注意一下 Time 列的信息，这是在极短的时间内发生的❷。当窗口大小像这样快速减小时，它很可能会减至零，如图 11-17 所示，数据包 4 正是这样的情形。

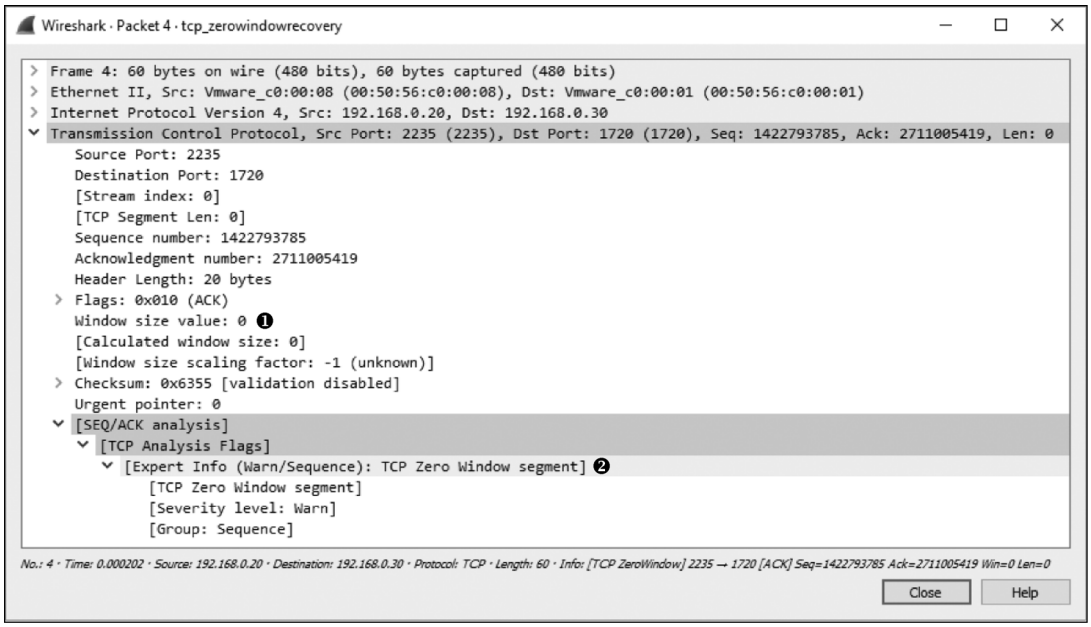


图 11-17　零窗口数据包说明了主机不能再接收任何数据

第 4 个数据包也是从 192.168.0.20 发往 192.168.0.30 的，但它的目的是告诉 192.168.0.30 它不能再接收任何数据。在 TCP 头部就可以看到这个数值 0❶，而且 Wireshark 也在 Packet List 面板的 Info 列以及 TCP 头部 SEQ/ACK Analysis 部分，告诉我们这是一个零窗口数据包❷。

异步社区Kensuke(18319066421) 专享 请尊重版权

一旦收到零窗口数据包，192.168.0.30 这个设备就不再发送任何数据，直到它从 192.168.0.20 收到一个窗口更新，通知它窗口大小已经增长了为止。幸好，在这个捕获文件里，导致零窗口的问题是暂时的。因此，如图 11-18 所示，发送的下一个数据包就是窗口更新。

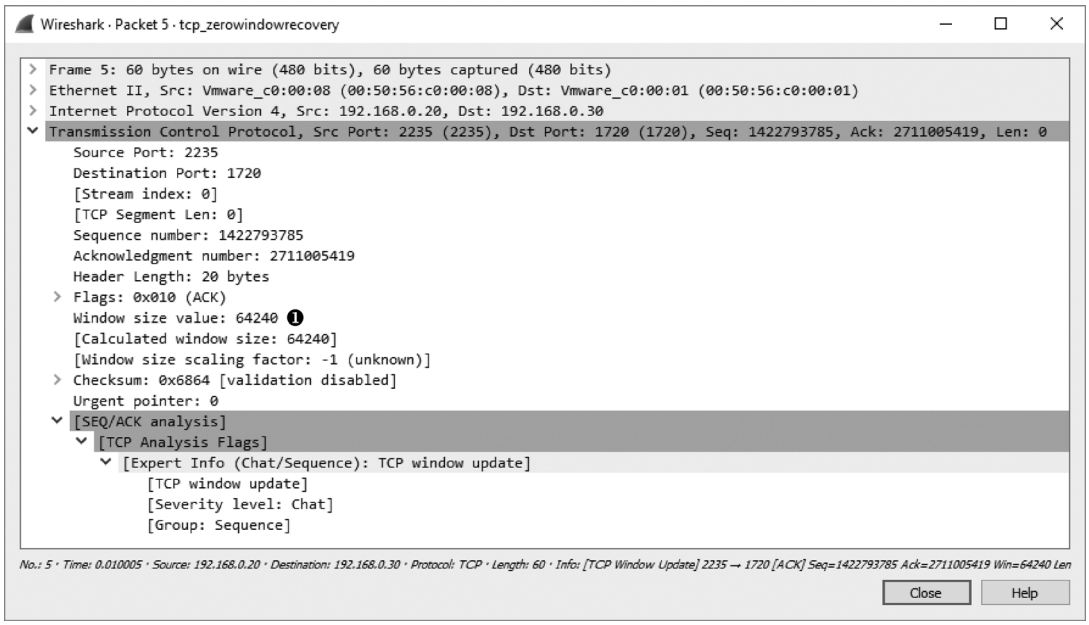


图 11-18　TCP 窗口更新数据包告诉其他主机它又可以传输数据了

在这个例子中，窗口大小增长到了非常健康的 64240 字节❶。Wireshark 再一次在 SEQ/ACK Analysis 标题下面告诉我们，这是一个窗口更新。

一旦收到这个更新数据包，192.168.0.30 主机就可以再次发送数据，如数据包 6 和 7 所示。这个过程非常迅速。就算它只是多持续一点点时间，也可能会引起网络“打嗝”，导致数据传输变慢或失败。

最后再看滑动窗口，查看一下 tcp_zerowindowdead.pcap 文件。捕获记录中的第一个数据包是从 195.81.202.68 发送到 172.31.136.85 的正常 HTTP 流量。如图 11-19 所示，紧接着就是一个从 172.31.136.85 返回的零窗口数据包。

这看起来跟图 11-17 里的零窗口数据包非常相似，但结果却很不相同。在图 11-20 中，我们并没有看到 172.31.136.85 主机发送使通信恢复的窗口更新，而是看到一个保活数据包。

异步社区Kensuke(18319066421) 专享 请尊重版权

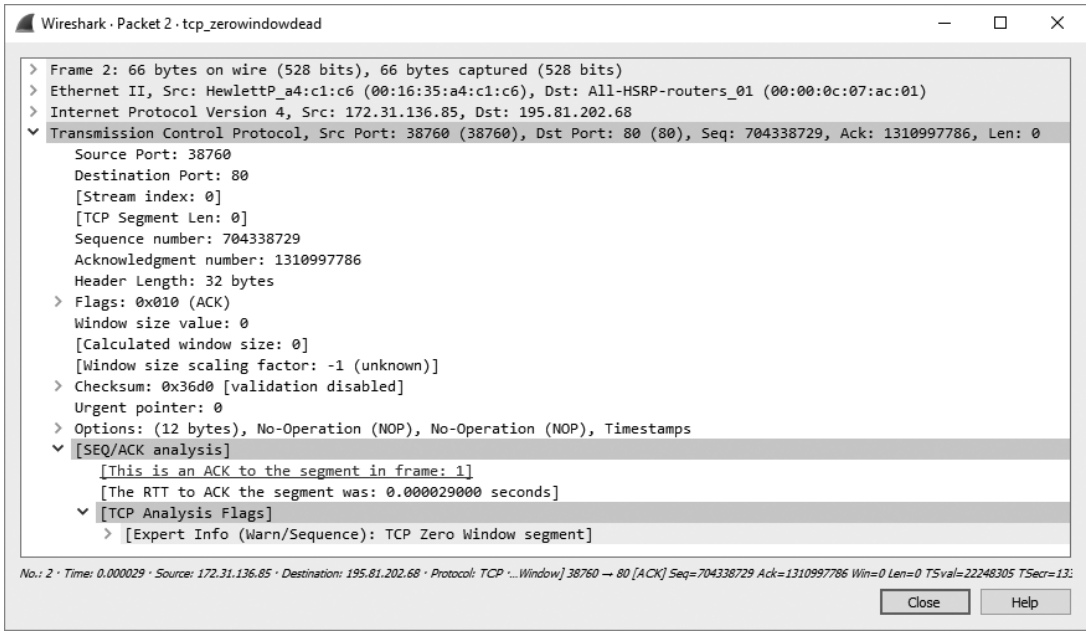


图 11-19　零窗口数据包使数据传输暂停

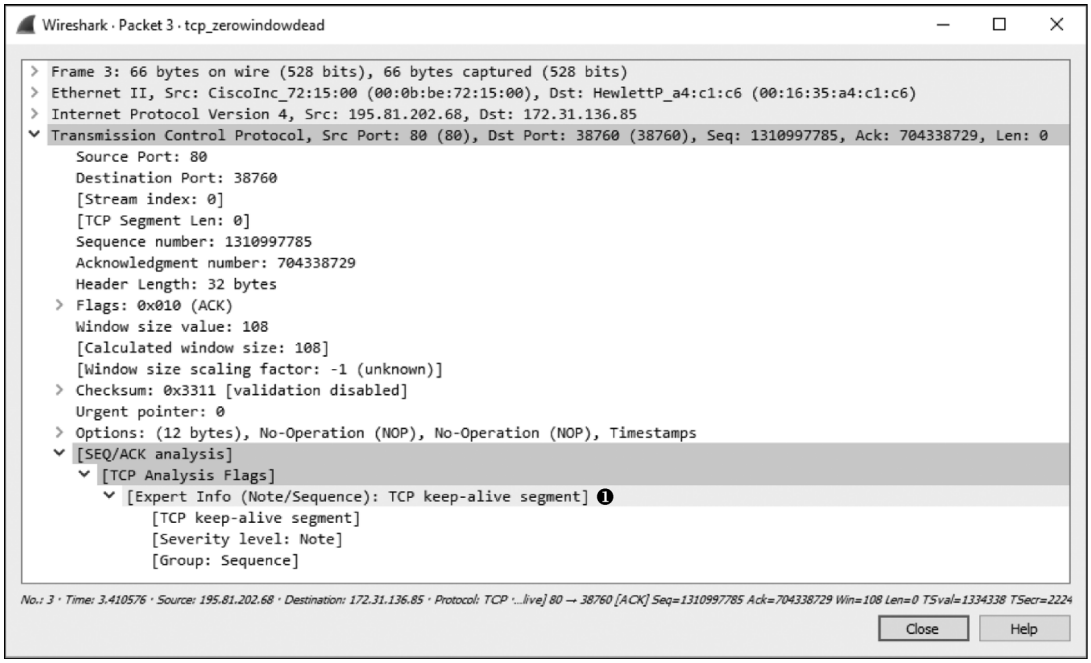


图 11-20　保活数据包保证零窗口主机仍然在线

异步社区Kensuke(18319066421) 专享 请尊重版权

Wireshark 在 Packet Details 面板中 TCP 头部的 SEQ/ACK Analysis 标题下，将这个数据包标记为保活数据包❶。我们从 Time 列可得知，在收到上一个数据包 3.4s 后，出现了这个数据包。如图 11-21 所示，这个过程又持续了几次：一台主机发送零窗口数据包，另一台则发送保活数据包。

No.	Time ❶	Source	Destination	Protocol	Length	Info
2	0.000029	172.31.136.85	195.81.202.68	TCP	66	[TCP ZeroWindow] 38760 → 80 [ACK] Seq=704338729 Ack=1310997786 Win=0 Len=0 TSv…
3	3.410576	195.81.202.68	172.31.136.85	TCP	66	[TCP Keep-Alive] 80 → 38760 [ACK] Seq=1310997785 Ack=704338729 Win=108 Len=0 T…
4	0.000031	172.31.136.85	195.81.202.68	TCP	66	[TCP ZeroWindow] 38760 → 80 [ACK] Seq=704338729 Ack=1310997786 Win=0 Len=0 TSv…
5	6.784127	195.81.202.68	172.31.136.85	TCP	66	[TCP Keep-Alive] 80 → 38760 [ACK] Seq=1310997785 Ack=704338729 Win=108 Len=0 T…
6	0.000029	172.31.136.85	195.81.202.68	TCP	66	[TCP ZeroWindow] 38760 → 80 [ACK] Seq=704338729 Ack=1310997786 Win=0 Len=0 TSv…
7	13.536714	195.81.202.68	172.31.136.85	TCP	66	[TCP Keep-Alive] 80 → 38760 [ACK] Seq=1310997785 Ack=704338729 Win=108 Len=0 T…
8	0.000047	172.31.136.85	195.81.202.68	TCP	66	[TCP ZeroWindow] 38760 → 80 [ACK] Seq=704338729 Ack=1310997786 Win=0 Len=0 TSv…

图 11-21　零窗口和保活数据包不断出现

这些保活数据包以 3.4s、6.8s、13.5s 的间隔出现❶。这个过程可能会持续相当长的时间，这取决于通信设备采用了哪个操作系统。在这个例子中，你可以发现，随着 Time 列数值的增长，连接暂停了将近 25s。想象一下，尝试向域控制器认证或者从网上下载文件时，25s 的延迟真是难以接受！

11.3　从 TCP 错误控制和流量控制中学到的

让我们具体看一下重传、重复 ACK 和滑动窗口机制。下面是处理延迟问题的一些注意事项。

1. 重传数据包

发生重传是因为客户端检测到服务器没有接收到它发送的数据。因此，能否看到重传取决于你在分析通信的哪一端。如果你从服务器那端捕获数据，并且它确实没有接收到客户端发送、重传的数据包，那你将被蒙在鼓里，因为你看不到重传的数据包。如果你怀疑服务器那端受到了数据包丢失的影响，（可能的话）你应该考虑在客户端那边捕获流量以观察是否有重传数据包。

2. 重复 ACK 数据包

我倾向于把重复 ACK 看作重传的表面对立词（pseudo-opposite），因为当服务器检测到来自客户端的数据包丢失时就发送它。在多数情况下，你可以在通信两端捕获到重复 ACK 的流量。记住，当接收到乱序数据包时才触发重复 ACK。例如，如果服务器只接收到了第 1 个和第 3 个数据包，就会发送一个重复 ACK

异步社区Kensuke(18319066421) 专享 请尊重版权

以引发第 2 个数据包的快速重传。由于你已经接收到第 1 个和第 3 个数据包，那不管什么情况导致第 2 个数据包丢失，都可能只是暂时的，因此在大部分情况下，你可以成功发送并接收重复 ACK。当然，有时候这个情况也未必成立。因此，当你在服务器端怀疑有数据包丢失却没有看到任何重复 ACK 时，可以考虑在通信的客户端捕获数据包。

3. 零窗口和保活数据包

滑动窗口与服务器接收、处理数据的故障直接相关。服务器的某些问题可以直接导致窗口大小减少或达到零窗口状态，因此如果你发现这样的情况，就应该集中调查那里。通常你会在网络通信的两端看见窗口更新数据包。

11.4 定位高延迟的原因

有时候，数据包丢失并不是延迟的原因。也许你会发现，当两台主机间通信很慢时，并没有 TCP 重传或重复 ACK 这样的常见特征。在这些情况下，你需要采用其他技术来定位高延迟的原因。

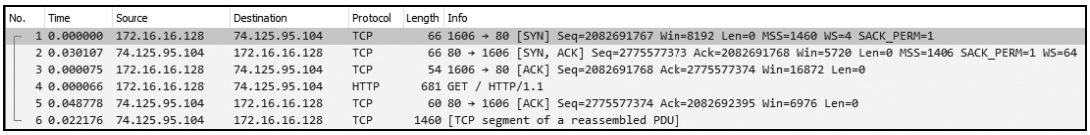
在这里，最有效的办法之一是查看初始连接握手以及接下来的两个数据包。例如，考虑客户端和 Web 服务器之间的一个简单连接，客户端尝试浏览托管在服务器上的一个站点。我们关注通信序列里的前 6 个数据包，包括 TCP 握手、初始 HTTP GET 请求、对这个 GET 请求的确认，以及从服务器发往客户端的第一个数据包。

注意　为了跟紧本章节，请确保你已经在 Wireshark 中设置了恰当的时间显示格式。在 Wireshark 中选择 View -> Time Display Format -> Seconds Since Previous Displayed Packet。

11.4.1 正常通信

我们将在本章的稍后部分详细讨论网络基线。目前，只知道你需要一个正常通信的基线，并与高延迟的情况作比较。在这些例子中，我们将使用 latency1.pcap 文件。由于我们已经讨论过了 TCP 握手和 HTTP 通信的细节，因此在这里我们将跳过它们。实际上，我们根本不需要再看 Packet Details 面板。如图 11-22 所示，我们只关心 Time 列。

异步社区Kensuke(18319066421) 专享 请尊重版权



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.16.16.128	74.125.95.104	TCP	66	1606 → 80 [SYN] Seq=2082691767 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
2	0.030107	74.125.95.104	172.16.16.128	TCP	66	80 → 1606 [SYN, ACK] Seq=2775577373 Ack=2082691768 Win=5720 Len=0 MSS=1406 SACK_PERM=1 WS=64
3	0.000075	172.16.16.128	74.125.95.104	TCP	54	1606 → 80 [ACK] Seq=2082691768 Ack=2775577374 Win=16872 Len=0
4	0.000066	172.16.16.128	74.125.95.104	HTTP	681	GET / HTTP/1.1
5	0.048778	74.125.95.104	172.16.16.128	TCP	60	80 → 1606 [ACK] Seq=2775577374 Ack=2082692395 Win=6976 Len=0
6	0.022176	74.125.95.104	172.16.16.128	TCP	1460	[TCP segment of a reassembled PDU]

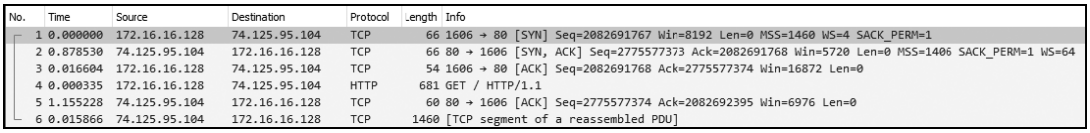
图 11-22　这些流量发生得相当快，被认为是正常的

这个通信序列是相当快的，全过程花了不到 0.1s。

接下来我们查看的几个捕获文件将包含相同的流量模式，只是在数据包时序上有些许不同。

11.4.2　慢速通信——线路延迟

现在我们转向捕获文件 latency2.pcap。如图 11-23 所示，注意，除了时间值之外，所有数据包都和上一个文件中的相同。



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.16.16.128	74.125.95.104	TCP	66	1606 → 80 [SYN] Seq=2082691767 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
2	0.878530	74.125.95.104	172.16.16.128	TCP	66	80 → 1606 [SYN, ACK] Seq=2775577373 Ack=2082691768 Win=5720 Len=0 MSS=1406 SACK_PERM=1 WS=64
3	0.016604	172.16.16.128	74.125.95.104	TCP	54	1606 → 80 [ACK] Seq=2082691768 Ack=2775577374 Win=16872 Len=0
4	0.000335	172.16.16.128	74.125.95.104	HTTP	681	GET / HTTP/1.1
5	1.155228	74.125.95.104	172.16.16.128	TCP	60	80 → 1606 [ACK] Seq=2775577374 Ack=2082692395 Win=6976 Len=0
6	0.015866	74.125.95.104	172.16.16.128	TCP	1460	[TCP segment of a reassembled PDU]

图 11-23　数据包 2 和 5 有很高的延迟

当我们开始逐个查看这 6 个数据包时，会很快遇到延迟的第一个标志。客户端（172.16.16.128）发送初始 SYN 数据包，开始 TCP 握手。在收到服务端（74.125.95.104）返回的 SYN/ACK 数据包之前有 0.87s 的延迟。这是我们受到线路延迟影响的第一个迹象，这是客户端和服务器之间的设备导致的。

由于数据包传输的特性，我们可以确定这是线路延迟的问题。当服务器收到一个 SYN 数据包时，由于不涉及任何传输层以上的处理，因此发送一个响应只需要非常小的处理量。即使服务器正承受巨大的流量负载，通常它也会迅速地向 SYN 数据包响应一个 SYN/ACK。这排除了服务器导致高延迟的可能性。

客户端的可能性也被排除了，因为它在此时除了接收 SYN/ACK 数据包以外什么也没干。排除了客户端和服务器的原因，那么网络缓慢的原因应该在捕获记录的前两个数据包里。

继续看，我们发现完成三次握手的 ACK 数据包传输很快，客户端发送的

异步社区Kensuke(18319066421) 专享 请尊重版权

HTTP GET 请求也同样如此。产生这两个数据包的处理过程是收到 SYN/ACK 后在客户端本地发生的，所以只要客户端没有沉重的处理负载，这两个数据包应该立刻就能发出去。

在数据包 5，我们看到它的时间值也高得令人难以置信。看来，我们发送初始 HTTP GET 请求之后，经过 1.15s 才收到从服务器返回的 ACK 数据包。收到 HTTP GET 请求后，服务器在发送数据之前先发送了一个 TCP ACK，同样这也不需要服务器耗费太多处理资源。这是线路延迟的另一个标志。

每次遇上线路延迟，你几乎都会在通信过程中初始握手的 SYN/ACK 以及其他 ACK 数据包中看到这样的情景。虽然这个信息并没有告诉你网络高延迟的确切原因，但它起码告诉你不是客户端或服务器的问题，所以你能意识到延迟是因为中间的一些设备出了问题。此刻，你可以开始检查受影响主机之间的防火墙、路由器、代理服务器等设备，以确定问题所在。

11.4.3 通信缓慢——客户端延迟

我们查看的下一个延迟情景包含在 latency3.pcap 文件中，如图 11-24 所示。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.16.16.128	74.125.95.104	TCP	66	1606 → 80 [SYN] Seq=2082691767 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
2	0.023790	74.125.95.104	172.16.16.128	TCP	66	80 → 1606 [SYN, ACK] Seq=2775577373 Ack=2082691768 Win=5720 Len=0 MSS=1406 SACK_PERM=1 WS=64
3	0.014894	172.16.16.128	74.125.95.104	TCP	54	1606 → 80 [ACK] Seq=2082691768 Ack=2775577374 Win=16872 Len=0
4	1.345023	172.16.16.128	74.125.95.104	HTTP	681	GET / HTTP/1.1
5	0.046121	74.125.95.104	172.16.16.128	TCP	60	80 → 1606 [ACK] Seq=2775577374 Ack=2082692395 Win=6976 Len=0
6	0.016182	74.125.95.104	172.16.16.128	TCP	1460	[TCP segment of a reassembled PDU]

图 11-24　这个捕获记录中的缓慢数据包是初始 HTTP GET 请求数据包

刚开始这个捕获记录很正常，迅速完成了 TCP 握手，没有任何延迟。但握手完成之后，数据包 4 的 HTTP GET 请求出现了问题。该数据包继上一个数据包有 1.34s 的延迟。

我们应该查看数据包 3 和 4 之间到底发生了什么，以找到延迟的原因。数据包 3 是 TCP 握手过程中客户端发往服务器的最后一个 ACK，数据包 4 则是客户端发往服务器的 GET 请求。它们的共同点在于都是客户端发送的，与服务器无关。发送完 ACK 之后本应该很快就发送 GET 请求，因为所有动作都是以客户端为中心的。

不幸的是，ACK 并没有快速切换到 GET 请求。创建和传输 GET 数据包确实需要应用层的处理，而这个处理过程的延迟表明客户端无法及时执行该动作。这意味着通信高延迟的根源在于客户端。

异步社区Kensuke(18319066421) 专享 请尊重版权

11.4.4 通信缓慢——服务器延迟

我们查看的最后一个延迟情景使用了 latency4.pcap 文件，如图 11-25 所示。这是服务器延迟的一个例子。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.16.16.128	74.125.95.104	TCP	66	1606 → 80 [SYN] Seq=2082691767 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
2	0.018583	74.125.95.104	172.16.16.128	TCP	66	80 → 1606 [SYN, ACK] Seq=2775577373 Ack=2082691768 Win=5720 Len=0 MSS=1406 SACK_PERM=1 WS=64
3	0.016197	172.16.16.128	74.125.95.104	TCP	54	1606 → 80 [ACK] Seq=2082691768 Ack=2775577374 Win=16872 Len=0
4	0.000172	172.16.16.128	74.125.95.104	HTTP	681	GET / HTTP/1.1
5	0.047936	74.125.95.104	172.16.16.128	TCP	60	80 → 1606 [ACK] Seq=2775577374 Ack=2082692395 Win=6976 Len=0
6	0.982983	74.125.95.104	172.16.16.128	TCP	1460	[TCP segment of a reassembled PDU]

图 11-25　直到最后一个数据包才表现出高延迟

在这个捕获记录中，两台主机间的 TCP 握手过程很快就顺利完成了，这是个很好的开端。下一对数据包带来了更好的消息，初始 GET 请求和响应的 ACK 数据包也很快传输完毕。直到最后一个数据包，我们才发现了高延迟的迹象。

第 6 个数据包是服务器响应客户端 GET 请求的第一个 HTTP 数据包，但它竟然在服务器为 GET 请求发送 TCP ACK 之后延迟了 0.98s。数据包 5 到 6 的切换情况与我们在上一个情景中看见的握手 ACK 与 GET 请求之间的切换很相似。然而，在这个例子中，服务器才是我们关注的焦点。

数据包 5 是服务器响应客户端 GET 请求的 ACK。一旦发送这个数据包，服务器就应该立即开始发送数据。这个数据包中的数据访问、打包和传输是由 HTTP 协议完成的，由于这是一个应用层协议，因此需要服务器作一些处理。延迟收到这个数据包表明服务器不能及时处理这个数据，最终把延迟的根源指向了它。

11.4.5 延迟定位框架

我们使用 6 个数据包成功地定位了从客户端到服务器的网络高延迟的原因。这些场景看起来也许有点复杂，但图 11-26 应该能帮你更快地解决延迟问题。这些原则几乎可应用于任何基于 TCP 的通信。

注意　我们还没有讨论 UDP 延迟。因为 UDP 的设计目标是快速但不可靠，所以它没有内置任何延迟检测并从中恢复的功能。相反，它依赖于应用层协议（和 ICMP）来解决数据可靠传输的问题。

异步社区Kensuke(18319066421) 专享 请尊重版权

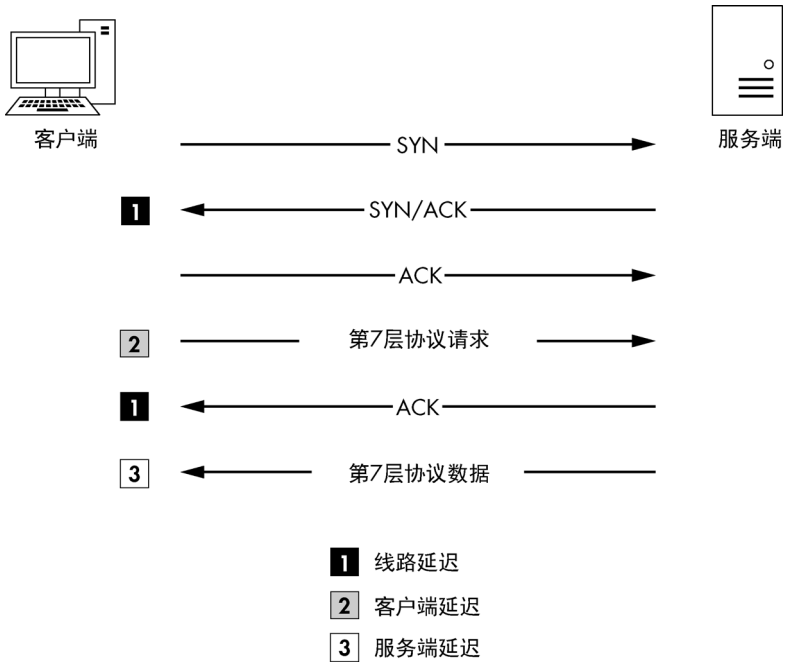


图 11-26　你可以用这张图解决自己的延迟问题

11.5　网络基线

当所有的努力都失败时，网络基线将成为检修网络缓慢故障最关键的数据之一。对我们的目的而言，网络基线包含来自网络不同端点的流量样本，包括大量我们认可的“正常”网络流量。网络基线的作用是在网络或设备工作不正常时作为比较的基准。

例如，考虑一个场景，网络上几个客户反映登录某个本地 Web 应用服务器时反应迟钝。如果你捕获这些流量并与网络基线对照，就会发现 Web 服务器一切正常，但嵌入到 Web 应用中的外部内容引发了额外的外部 DNS 请求，而这些请求比正常速度慢两倍。

也许不靠网络基线的帮助，你也能注意到异常的外部 DNS 服务器，但当你处理微妙的变化时，就不一定了。10 个 DNS 请求都比正常情况多耗费 0.1s，这跟一个 DNS 请求比正常多耗费 1s 一样糟糕，但没有网络基线的话，检测前者要难得多。

异步社区Kensuke(18319066421) 专享 请尊重版权

由于网络各不相同，因此网络基线的组件将有很大差异。接下来的几节提供了网络基线组件的几个例子。你也许会发现所有这些条目都能应用到你的网络基础设施，或只是很少一部分能适用。不管怎样，你应该把你的基线中的每一个组件置入这 3 个基本基线目录中：站点、主机和应用程序。

11.5.1 站点基线

站点基线的目的是获得网络上每个物理站点的整体流量快照。理想情况下，这将是 WAN 内的每一个段。

这个基线应该包含以下几个组件。

1. 使用的协议

在网络边缘（路由器/防火墙）捕获网段上所有设备的流量时，请使用协议分层统计窗口（**Statistics->Protocol Hierarchy**）来查看所有设备的流量。然后，你可以对照查看是否缺少本应出现的协议，或者网络上是否出现了新的协议。你也可以在协议的基础上，用它来发现高于正常数量的特定类型的流量。

2. 广播流量

这包含网段上的一切广播流量。在站点内任一点监听都可以捕获所有广播流量，通过它可以了解正常情况下谁将大量广播流量发送到网络中，这样你将很快确定是否出现了过多（或过少）的广播流量。

3. 身份验证序列

这包括任意客户端到所有服务的身份验证过程的流量，比如动态目录、Web 应用程序，以及特定组织的软件。身份验证通常是服务运行缓慢的一个方面。通过基线你可以确定身份验证是否是通信缓慢的原因。

4. 数据传输率

这通常包括在网络上测量这个站点到其他站点的大量数据传输。你可以使用 Wireshark 的捕获概述和绘图功能来确定传输速率和连接的一致性。这可能是你的一个很重要的站点基线。每当在网段上建立或拆除连接速度很慢时，你就可以运行与基线同样的数据传输并比较结果。这会告诉你连接是否真的很慢，甚至可能帮助你查找缓慢的原因。

异步社区Kensuke(18319066421) 专享 请尊重版权

11.5.2 主机基线

使用主机基线并不意味着你必须要在网络上测试每台主机。主机基线只需要在高流量或关键任务服务器上执行。基本上，一旦某台服务器运行缓慢，便会招来管理层愤怒的电话，你应当在那台主机上建立基线。

主机基线包含以下几个组件。

1. 使用的协议

当捕获这台主机的流量时，这个基线提供了使用协议分层统计窗口的好机会。然后，你可以对照看一看是否缺少本应出现的协议，或者主机上是否出现了新的协议。你也可以在协议的基础上，用它来发现高于正常数量的特定类型的流量。

2. 空闲/繁忙流量

这个基线只是简单包含了高峰和非高峰时段正常操作流量的总体捕获记录。了解到一天之中不同时段的连接数量及其占用的带宽大小，有助于你确定缓慢是用户负载还是其他原因造成的。

3. 启动/关闭

为了获取这个基线，你需要在主机启动和关闭时创建一个流量捕获记录。一旦计算机不能启动、不能关闭，或这两个过程异常缓慢，你就可以使用它确认问题是否跟网络有关。

4. 身份验证序列

这个基线要求在主机上捕获所有服务的身份验证过程的流量。身份验证通常是服务运行缓慢的一个方面。通过基线你可以确认身份验证是否是通信缓慢的原因。

5. 关联/依赖

这个基线需要持续更长时间的捕获，以确定这台主机依赖于哪些主机（以及哪些主机依赖于这台主机）。你可以通过会话窗口（**Statistics->Conversations**）查看这些关联和依赖。Web 服务器依赖于 SQL 服务器便是这样的例子。有时我们会意识不到主机间的一些潜在依赖关系，这时主机基线便能派上用场。通过这个，你可以确定主机不能正常运转，是因为配置错误还是因为所依赖主机的

异步社区Kensuke(18319066421) 专享 请尊重版权

高延迟。

11.5.3 应用程序基线

最后一个网络基线类别是应用程序基线。这个基线应该用于所有基于网络的关键业务应用程序。

应用程序基线包含以下几个组件。

1. 使用的协议

我们在这个基线中再次使用了 Wireshark 的协议分层统计窗口，但这次是在运行应用程序的主机上捕获流量。然后，你可以通过比较这个列表，发现依赖于这些协议的应用程序是否正常运转。

2. 启动/关闭

这个基线需要捕获应用程序启动和关闭时生成的流量。一旦应用程序不能启动、不能关闭，或这两个过程都异常缓慢，你就可以使用它确定原因。

3. 关联/依赖

这个基线需要持续更久的捕获，以通过会话窗口确定这个应用程序依赖的其他主机和应用程序。有时我们会意识不到应用程序间的一些潜在依赖关系，这时应用程序基线便能派上用场。通过这个，你可以确定应用程序不能正常运转，是因为配置错误还是因为所依赖应用程序的高延迟。

4. 数据传输率

你可以在应用程序服务器正常运转期间，使用 Wireshark 的捕获概述和绘图功能确定数据传输率和连接一致性。每当有人报告应用程序缓慢时，你就可以使用这个基线来确定当前问题是否是高利用率或高用户负载造成的。

11.5.4 基线的其他注意事项

下面是创建网络基线的一些额外注意事项。

- 创建每个基线都至少要经过 3 次：低流量期间一次（早晨）、高流量期间一次（下午三点左右）、无流量期间一次（深夜）。
- 有可能的话，尽量避免直接在需要创建基线的主机上捕获流量。因为在高流量期间，这可能会增加设备负载和影响性能，并可能因数据包丢失导致

异步社区Kensuke(18319066421) 专享 请尊重版权

基线无效。

- 你的基线可能包含一些与网络有关的私密信息，一定要保护好它。将它存储在安全的地方，只有合适的人才有访问权限。但同时别放得太偏，以免需要时找不到。可以考虑将它存放在 U 盘或者加密分区里。
- 让所有.pcap 文件与你的基线关联，为更常见的参考值写一份“小抄”，比如关联关系或数据传输率。

11.6 小结

本章内容聚焦于如何解决慢速网络的问题。我们讲述了 TCP 的一些有用的可靠性检测和恢复功能，演示了如何定位网络通信高延迟的原因，并讨论了网络基线的重要性以及它的一些组件。使用这里讨论的技术，再加上 Wireshark 的绘图和分析功能（在第 5 章讨论过），当你再接到电话抱怨网速很慢时，应该能应付自如。

异步社区Kensuke(18319066421) 专享 请尊重版权

第12章

安全领域的数据包分析



虽然本书主要集中于如何使用数据包分析技术解决网络故障，但在现实世界中，很多数据包分析工作都是为了解决安全问题。当入侵分析师检查来自可疑入侵者的网络流量，或取证人员调查恶意软件在主机上的感染程度时，就会用到数据包分析的方法。面向安全的数据包分析是一个很大的话题，都可以另写一本书了，本章只是带你尝尝鲜。

在本章，我们将扮演一位安全从业者，学习在网络层分析“肉鸡”[1]系统的各个方面。我们将涉及网络侦察、恶意的流量重定向，以及系统漏洞利用。接着，我们将扮演一位入侵分析师，剖析来自入侵检测系统的警报流量。即使你不是安全从业者，通过阅读本章，你也可以获得一些对网络安全的关键洞察。

[1] 被攻击者控制的机器。——译者注

异步社区Kensuke(18319066421) 专享 请尊重版权

12.1 网络侦察

攻击者采取的第一步行动是深入研究目标系统。这一步又叫“网络踩点”，通常使用各式各样的公开资源来完成，比如目标公司的主页或者 Google。这个研究完成后，攻击者通常开始扫描目标 IP 地址（或者域名）的开放端口或运行服务。

通过扫描，攻击者可以确定目标是否在线并且可达。例如，想象一下，一位银行大盗盯上了位于缅因街 123 号的目标——本市规模很大的银行。他花费数星期时间精心策划此次抢劫，却在抵达目的地后才发现银行已经搬迁到了万安街 555 号。还可以想象一个更糟糕的场景，劫匪计划在正常上班时间步行进入银行，以便对金库下手，刚到银行门口却发现今天歇业。确保目标在线并且可达是我们必须要解决的一个问题。

扫描的另一个重要收获是，它告诉了攻击者目标开放了哪些端口。回到我们刚才类比的银行劫匪，想一想，如果劫匪出现在银行门口，却对整幢楼的布局一无所知，会怎么样？他无法进入大楼，因为他不知道物理防御的弱点在哪里。

在本节中，我们会讨论如何用一些典型的扫描技术识别主机和它们开放的端口号，以及网络上的漏洞。

注意 到目前为止，本书说的“连接两端”都是指发送者和接收者，或者客户端和服务器。而本章提到的“连接两端”却是指攻击者或受害者。

12.1.1 SYN 扫描

首先对系统作 TCP SYN 扫描，又称为隐秘扫描或半开扫描。SYN 扫描是一种常见的扫描类型，有以下几个原因。

- 快速可靠。
- 在所有平台上都很准确，与 TCP 协议栈的实现无关。
- 比其他扫描技术更安静，不容易被发现。

TCP SYN 扫描依赖于三步握手过程，可以确定目标主机的哪些端口是开/

异步社区Kensuke(18319066421) 专享 请尊重版权

的。攻击者发送 TCP SYN 数据包到受害者的一定范围的端口上，就像要在这些端口上建立用于正常通信的连接似的。如图 12-1 所示，一旦受害者收到这个数据包，就可能会做出某些响应。

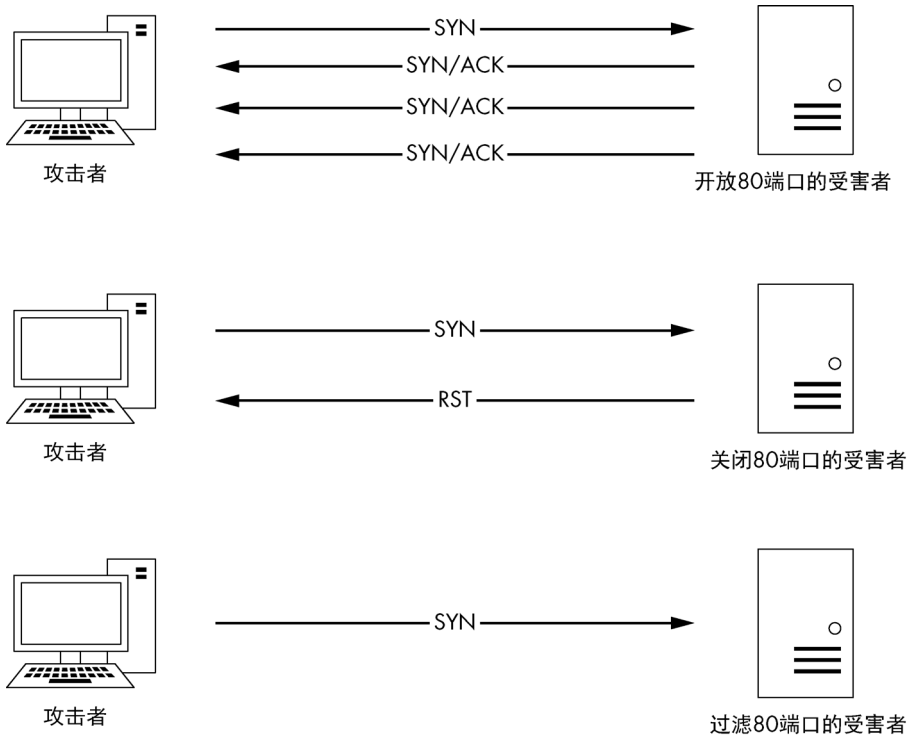


图 12-1　一次 TCP SYN 扫描的结果

如果受害者机器上某个服务正在监听的端口收到了 SYN 数据包，那么它将向攻击者回复一个 TCP SYN/ACK 数据包，也就是 TCP 握手的第二部分。这样攻击者就能知道这个端口是开放的，并且有一个服务在上面监听。正常情况下会发送一个 TCP ACK 包以完成连接握手，但此刻攻击者并不想这样，因为他还不想与主机通信。所以，攻击者并不打算完成 TCP 握手。

如果没有服务在被扫描的端口上监听，那么攻击者就收不到 SYN/ACK。按照受害者操作系统的不同配置，攻击者可能会收到响应的 RST 数据包，表示端口关闭了，或者，攻击者看不到任何响应。这意味着端口被某个中间设备过滤了，或许是防火墙，或许是主机本身。另一方面，也有可能是因为响应数据包在传输过程中丢失了。这个结果通常表明端口是关闭的，但说服力并不强。

异步社区Kensuke(18319066421) 专享 请尊重版权

捕获文件 synscan.pcp 提供了用 Nmap 工具进行 SYN 扫描的绝佳例子。Nmap 是 Fyodor 创立的一款稳定的网络扫描程序。它可以执行你能想到的任何一种扫描方式。

我们捕获的样本大概包含 2000 个数据包，说明这种扫描有一定的规模。确定这个扫描范围大小的最好办法之一就是查看 Conversations 窗口，如图 12-2 所示。在这里，你会看到攻击者（172.16.0.8）和受害者（63.13.134.52）之间只有一个 IPv4 会话❶。你也会看到，那里有 1994 个 TCP 会话❷——通信基本上是每一个端口对应一个新会话。

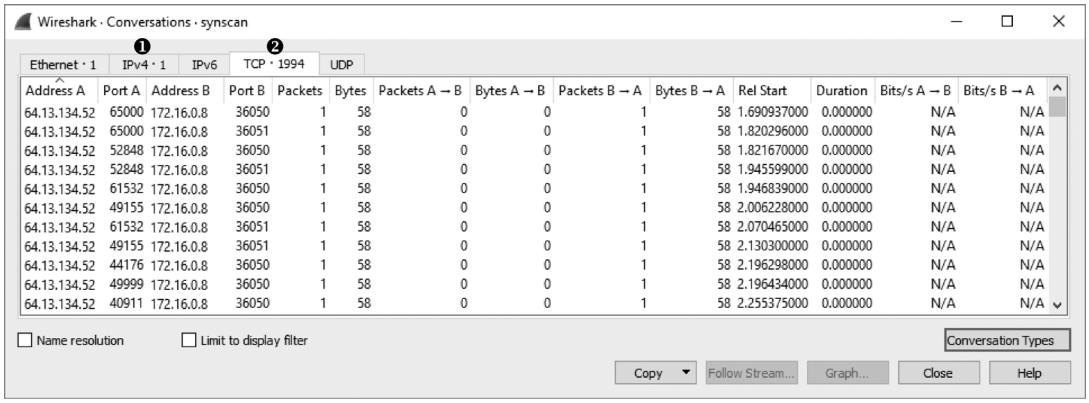


图 12-2　Conversations 窗口显示了正在进行的 TCP 通信

扫描是在极短时间内完成的，因此在捕获文件上滚动鼠标并不是寻找 SYN 数据包响应的好办法。在接收到响应之前，已经发送了更多的 SYN 数据包。幸好，我们可以创建过滤器，来帮助我们寻找正确的流量。

1. 在 SYN 扫描中使用过滤器

举一个筛选的例子。让我们看一看第 1 个数据包——发送到受害者 443 端口（HTTPS）的 SYN 数据包。为了查看是否有对这个数据包的响应，我们可以创建一个过滤器，以显示所有源端口或目标端口为 443 的流量。下面是如何快速设置的方法。

（1）在捕获文件中选择第一个数据包。

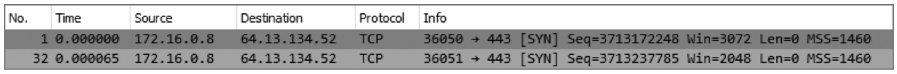
（2）在 Packet Details 面板中展开 TCP 头部。

（3）右键单击 Destination Port 字段，选择 Prepare as Filter，单击 Selected。

异步社区Kensuke(18319066421) 专享 请尊重版权

（4）这将在 filter 对话框放置一个过滤器，针对所有目标端口为 443 的数据包。现在，由于我们也需要源端口为 443 的数据包，所以点击屏幕顶端的 filter 栏，并删除过滤器的 dst 部分。

结果过滤器给出了两个数据包，都是攻击者发给受害者的 TCP SYN 数据包，如图 12-3 所示。

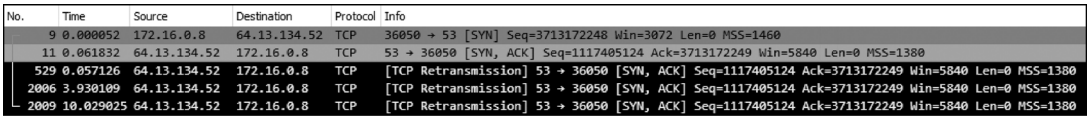


No.	Time	Source	Destination	Protocol	Info
1	0.000000	172.16.0.8	64.13.134.52	TCP	36050 → 443 [SYN] Seq=3713172248 Win=3072 Len=0 MSS=1460
32	0.000065	172.16.0.8	64.13.134.52	TCP	36051 → 443 [SYN] Seq=3713237785 Win=2048 Len=0 MSS=1460

图 12-3　两次尝试用 SYN 数据包建立连接

两个数据包都没有得到响应，有可能是因为响应数据包被受害者主机或中间设备过滤了，或者端口是关闭的。但最终来说，对 443 端口的扫描结果是不确定的。

我们可以用同样的技术来分析其他数据包，看一看有没有不同的结果。首先，单击过滤器旁边的 Clear 按钮，清空之前创建的过滤器。然后选择列表中的第 9 个数据包。这是目标端口为 53 的 SYN 数据包，通常与 DNS 有关。使用前面提到的方法，创建一个基于目标端口的过滤器，并删除 dst 部分，这样它就应用到所有与 TCP 53 端口有关的流量了。当使用这个过滤器时，你会看见 5 个数据包，如图 12-4 所示。



No.	Time	Source	Destination	Protocol	Info
9	0.000052	172.16.0.8	64.13.134.52	TCP	36050 → 53 [SYN] Seq=3713172248 Win=3072 Len=0 MSS=1460
11	0.061832	64.13.134.52	172.16.0.8	TCP	53 → 36050 [SYN, ACK] Seq=1117405124 Ack=3713172249 Win=5840 Len=0 MSS=1380
529	0.057126	64.13.134.52	172.16.0.8	TCP	[TCP Retransmission] 53 → 36050 [SYN, ACK] Seq=1117405124 Ack=3713172249 Win=5840 Len=0 MSS=1380
2006	3.930109	64.13.134.52	172.16.0.8	TCP	[TCP Retransmission] 53 → 36050 [SYN, ACK] Seq=1117405124 Ack=3713172249 Win=5840 Len=0 MSS=1380
2009	10.029025	64.13.134.52	172.16.0.8	TCP	[TCP Retransmission] 53 → 36050 [SYN, ACK] Seq=1117405124 Ack=3713172249 Win=5840 Len=0 MSS=1380

图 12-4　表明端口是开放的 5 个数据包

第 1 个是我们在捕获之初选择的 SYN 数据包。第 2 个则是来自受害者的响应。这是一个 TCP SYN/ACK 数据包——实施三次握手时期待的响应。在正常情况下，下一个数据包应该是发送初始 SYN 的主机发送的 ACK。然而，在这个例子中，攻击者并不想建立连接，因而没有发送响应。受害者重传了 3 次 SYN/ACK 包才放弃。由于尝试与主机的 53 端口通信时收到了 SYN/ACK 响应，因此我们可以确定有一个服务在监听该端口。

让我们在数据包 13 上再次重复此过程。这是一个目标端口为 113 的 SYN 数据包，通常与 Ident 协议有关，此协议常用于 IRC 的身份识别和验证服务。如果你在这个数据包上使用同一类型的过滤器，就会发现 4 个数据包，如图 12-5 所示。

异步社区Kensuke(18319066421) 专享 请尊重版权

No.	Time	Source	Destination	Protocol	Info
13	0.000070	172.16.0.8	64.13.134.52	TCP	36050 → 113 [SYN] Seq=3713172248 Win=4096 Len=0 MSS=1460
14	0.061491	64.13.134.52	172.16.0.8	TCP	113 → 36050 [RST, ACK] Seq=2462244745 Ack=3713172249 Win=0 Len=0
530	0.006942	172.16.0.8	64.13.134.52	TCP	36061 → 113 [SYN] Seq=3696394776 Win=2048 Len=0 MSS=1460
571	0.000827	64.13.134.52	172.16.0.8	TCP	113 → 36061 [RST, ACK] Seq=1027049353 Ack=3696394777 Win=0 Len=0

图 12-5　SYN 之后紧随一个 RST，表明端口是关闭的

第 1 个数据包是初始 SYN，紧接着是来自受害者的 RST。这是受害者目标端口不接受连接的迹象，表明很可能没有服务运行在上面。

2. 识别开放和关闭的端口

理解了 SYN 扫描能引起的不同响应类型后，自然而然会想到去找一个方法——如何快速识别哪些端口是开放的还是关闭的。答案再次落到了 Conversations 窗口内。在这个窗口中，你可以通过数据包数量排序 TCP 会话，单击 Packets 列直到箭头向下就可以让最高值靠前，如图 12-6 所示。

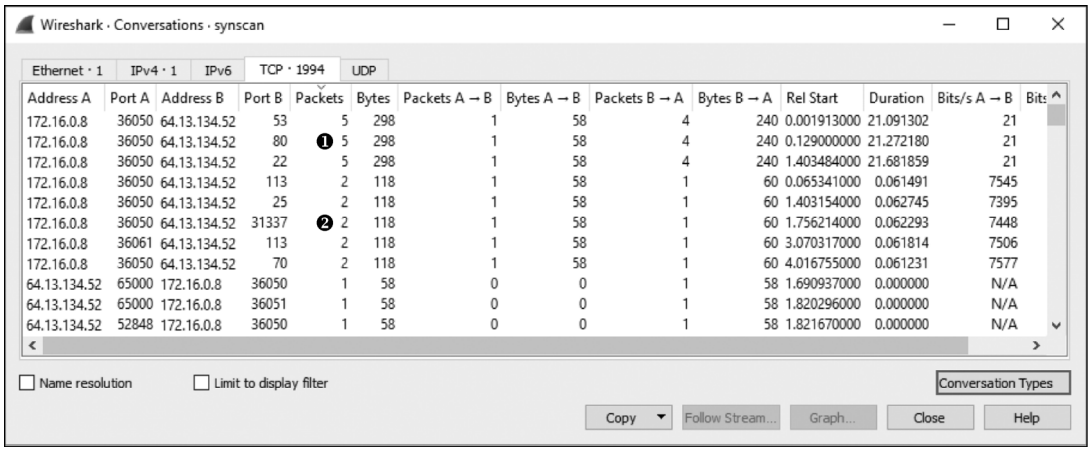


图 12-6　用 Conversations 窗口寻找开放端口

3 个被扫描的端口在各自会话中包含 5 个数据包❶。我们知道 53、80 和 22 端口是开放的，因为这 5 个数据包表示初始 SYN、来自受害者的 SYN/ACK 及其 3 次重传。

有 5 个端口的通信只包含 2 个数据包❷。第 1 个是初始 SYN，第 2 个是来自受害者的 RST。这表明 113、25、31337、113 和 70 端口是关闭的。

Conversations 窗口剩下的项只包含 1 个数据包，意味着受害者主机并没有响应初始 SYN 包。剩下的这些端口很可能是关闭的，但我们不能确定。

异步社区Kensuke(18319066421) 专享 请尊重版权

12.1.2 操作系统指纹

了解目标的操作系统对攻击者有极大价值。了解到目标使用的操作系统，将确保攻击者实施具有针对性的攻击手段。同时这个信息也有助于攻击者成功进入系统后，在目标文件系统找到关键文件和目录。

“操作系统指纹术”是指在没有物理接触的情况下，用来确定机器运行的操作系统的一组技术。操作系统指纹术分为两种类型：被动式和主动式。

1. 被动式指纹技术

被动式指纹技术通过分析目标发送的数据包的某些字段来确定目标使用的操作系统。这项技术之所以称为“被动”，是因为你只监听目标主机发送的数据包，但并不主动向目标发送任何流量。对黑客来说，这是理想的操作系统指纹技术，因为它非常隐蔽。

也就是说，我们只需要基于目标主机发送的数据包，就能确定它用的是哪一种操作系统了？嗯，这其实非常容易。由于 RFC 文件定义的协议并未规定全部技术参数的值，这完全是有可能的。虽然 TCP、UDP 和 IP 头部的各个字段都有特定含义，但并没有定义这些字段的默认值。这意味着不同操作系统实现的 TCP/IP 协议栈都必须为这些字段定义它自己的默认值。表 12-1 列出了一些与操作系统实现有关的常见字段及其默认值。

表 12-1　常见的被动式指纹值

协议头部	字段	默认值	操作系统
IP	初始 TTL	64	Nmap、BSD、Mac OS 10、Linux
		128	Novell、Windows
		255	Cisco IOS、Palm OS、Solaris
IP	不分片标志	Set	BSD、Mac OS 10、Linux、Novell、Windows、Palm OS、Solaris
		Not set	Nmap、Cisco IOS
TCP	最长段大小	0	Nmap
		1440	Windows、Novell
		1460	BSD、Mac OS 10、Linux、Solaris
TCP	窗口大小	1024-4096	Nmap
		65535	BSD、Mac OS 10
		2920-5840	Linux
		16384	Novell

异步社区Kensuke(18319066421) 专享 请尊重版权

续表

协议头部	字段	默认值	操作系统
TCP	SackOK	4128	Cisco IOS
		24820	Solaris
		可变	Windows
		设置	Linux、Windows、OpenBSD
		不设置	Nmap、FreeBSD、Mac OS 10、Novell、Cisco IOS、Solaris

捕获文件 passiveosfingerprinting.pcap 中的数据包是这项技术的绝佳例子。该文件含有两个数据包。它们都是目标端口为 80 的 TCP SYN 数据包，但来自不同的主机。仅仅使用这些数据包中的值，并参考表 12-1，我们就能确定每台主机使用的操作系统架构。每一个数据包的细节如图 12-7 所示。

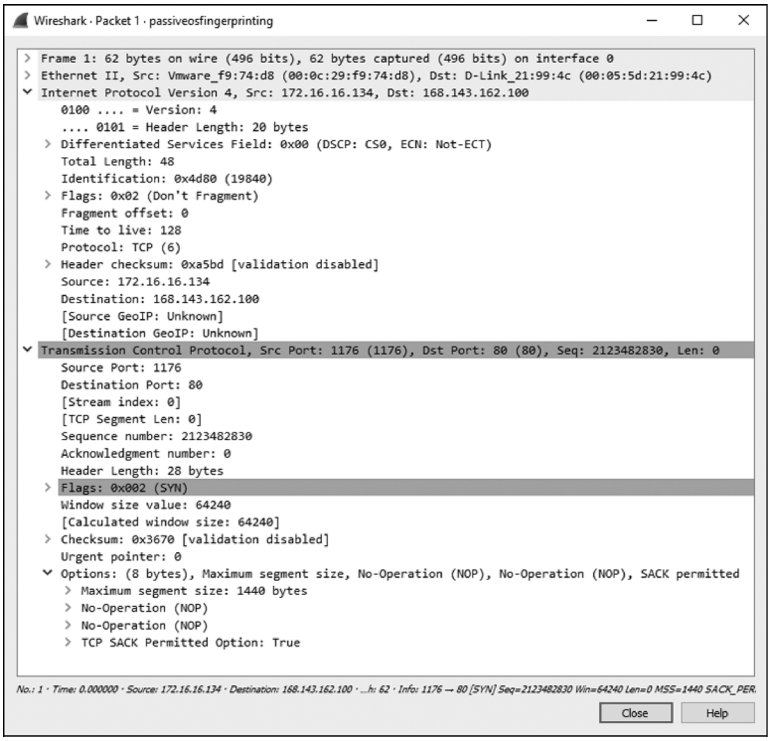


图 12-7　这些数据包可以告诉我们它们来自哪种操作系统

异步社区Kensuke(18319066421) 专享 请尊重版权

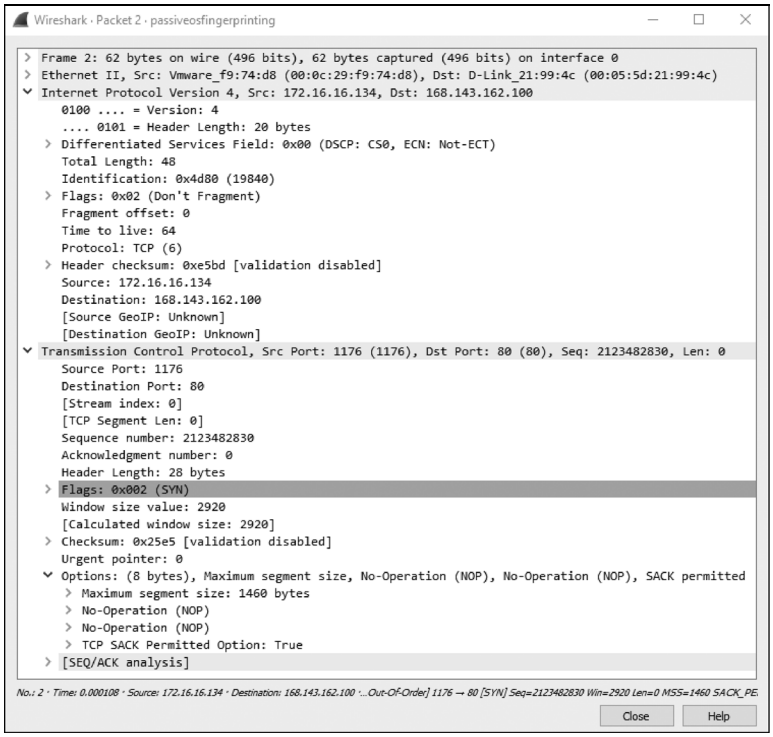


图 12-7　这些数据包可以告诉我们它们来自哪种操作系统（续）

参考表 12-1，我们将这些数据包的相关字段分类，创建了表 12-2。

表 12-2　操作系统指纹术的数据包分类

协议头部	字段	数据包 1 的值	数据包 2 的值
IP	初始 TTL	128	64
IP	不分片标志	设置/不设置	设置/不设置
TCP	最长段大小	1440 Bytes	1460 Bytes
TCP	窗口大小	64240 Bytes	2920 Bytes
TCP	SackOK	设置/不设置	设置/不设置

基于这些值，我们可以得出结论：发送数据包 1 的设备运行 Windows 的可能性最大，而发送数据包 2 的设备运行 Linux 的可能性最大。

要记住，表 12-2 列出的被动式操作系统指纹技术的常见识别字段并不完整。很多实现上的“怪癖”可能会导致真实值与期望值的偏差。所以，你不能完全依赖被动式操作系统指纹技术得到的结果。

异步社区Kensuke(18319066421) 专享 请尊重版权

注意　有一个叫 p0f 的工具使用了操作系统指纹识别技术。该工具分析捕获数据包的相关字段，然后输出可能的操作系统。使用像 p0f 这样的工具，你不仅能了解到操作系统架构，有时甚至能了解适当的版本号或者补丁级别。

2. 主动式指纹技术

当被动监听流量不能得出想要的结果时，可能需要一个更直接的方法。这种方法叫做“主动式指纹技术”。它是指攻击者主动向受害者发送特意构造的数据包以引起响应，然后从响应数据包中获知受害者机器操作系统的技术。当然，由于这种方法要与受害者直接通信，因此它并不是最隐蔽的，但它可以做到非常高效。

捕获文件 activeosfingerprinting.pcap 包含了使用 Nmap 扫描工具发起主动式指纹扫描的例子。文件中有一些是 Nmap 发送的探测数据包，这些探测数据包引起的响应可用于识别操作系统。Nmap 记录下对这些探测数据包的响应并创建一个指纹，与指纹数据库对比后得出结论。

注意　Nmap 使用的主动式操作系统指纹技术十分复杂。想了解 Nmap 如何执行主动式操作系统指纹技术，请阅读 Nmap 的权威指南——*Nmap Network Scanning*。这是 Nmap 扫描器作者 Gordon “Fyodor” Lyon 的著作。

12.2 流量操纵

12.2.1 ARP 缓存污染攻击

在第 7 章中，我们讨论了 ARP 协议是如何将网络中的 IP 地址映射成 MAC 地址的，在第 2 章中，我们讨论了将 ARP 缓存污染攻击作为监听主机流量的方法。ARP 缓存污染攻击是网络工程师高效实用的工具。然而，若有恶意企图，它也是一个非常致命的中间人攻击（man-in-the-middle，MITM）方法。

在 MITM 攻击中，攻击者重定向两台主机间的流量，试图在传输过程中拦截或修改。MITM 攻击有多种形式，包括会话劫持、DNS 欺骗，以及 SSL 劫持。

异步社区Kensuke(18319066421) 专享 请尊重版权

ARP 缓存污染攻击之所以有效，是因为特意构造的 ARP 数据包使两台主机相信它们在互相通信，而实际上它们却是与一个在中间转发数据包的第三方通信。

文件 arppoison.pcap 包含了 ARP 缓存污染攻击的一个例子。当打开它时，第一眼你会发现这些流量看起来很正常。然而，如果你跟进这些数据包，就会发现我们的受害者 172.16.0.107 在浏览 Google 并执行搜索。搜索的结果导致了一些 HTTP 流量，并夹杂一些 DNS 查询。

我们知道 ARP 缓存污染攻击是发生在第二层的技术，所以如果只是在 Packet List 面板里随意浏览，恐怕很难发现任何异常。因此，我们在 Packet List 面板里增加几列，过程如下。

（1）选择 **Edit->Preferences**。

（2）单击 Preferences 窗口左边的 **Columns**。

（3）单击 **Add**。

（4）输入 `Source MAC` 并按回车键。

（5）在 **Field type** 下拉列表里，选择 **Hw src addr (resolved)**。

（6）单击新增加的项，拖动它到 **Source** 列后面。

（7）单击 **Add**。

（8）输入 `Dest MAC` 并按回车键。

（9）在 **Field type** 下拉列表里，选择 **Hw dest addr (resolved)**。

（10）单击新增加的项，拖动它到 **Destination** 列后面。

（11）单击 **OK** 使改动生效。

当完成这些步骤时，你的屏幕应该跟图 12-8 一样。你现在应该有额外的两列，分别显示了数据包的来源 MAC 地址和目标 MAC 地址。

如果你还打开了 MAC 地址解析，应该会看到通信设备的 MAC 地址表明它是 Dell 或 Cisco 硬件。这是很重要的，因为当我们滚动整个捕获记录时，这些信息在数据包 54 就开始改变了。我们看到了一些奇怪的 ARP 流量，在 Dell 主机（受害者）和新出现的 HP 主机（攻击者）之间交互，如图 12-9 所示。

异步社区Kensuke(18319066421) 专享 请尊重版权

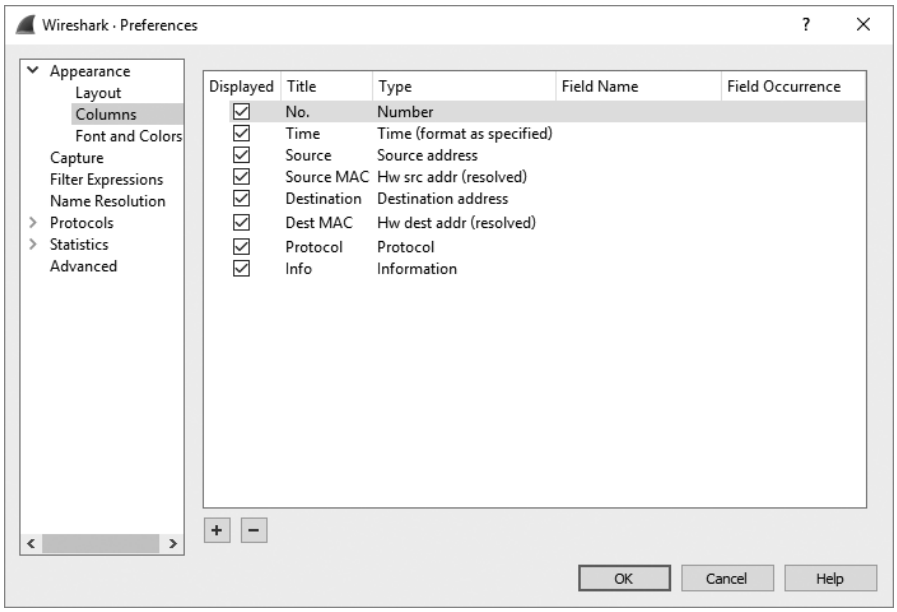
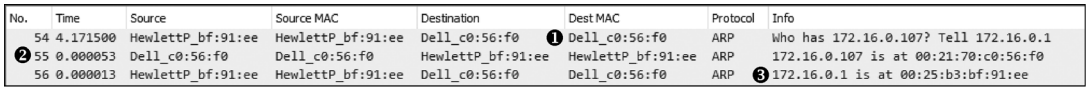


图 12-8　Column 配置屏幕，包含了新增的来源和目标硬件地址列



No.	Time	Source	Source MAC	Destination	Dest MAC	Protocol	Info
54	4.171500	HewlettP_bf:91:ee	HewlettP_bf:91:ee	Dell_c0:56:f0	❶ Dell_c0:56:f0	ARP	Who has 172.16.0.107? Tell 172.16.0.1
❷ 55	0.000053	Dell_c0:56:f0	Dell_c0:56:f0	HewlettP_bf:91:ee	HewlettP_bf:91:ee	ARP	172.16.0.107 is at 00:21:70:c0:56:f0
56	0.000013	HewlettP_bf:91:ee	HewlettP_bf:91:ee	Dell_c0:56:f0	Dell_c0:56:f0	ARP	❸ 172.16.0.1 is at 00:25:b3:bf:91:ee

图 12-9　Dell 设备和 HP 设备间奇怪的 ARP 流量

在进一步深入之前，注意一下这次通信中涉及的端点，由表 12-3 列出。

表 12-3　监视的端点

角色	设备类型	IP 地址	MAC 地址
受害者	Dell	172.16.0.107	00:21:70:c0:56:f0
路由器	Cisco	172.16.0.1	00:26:0b:21:07:33
攻击者	HP	未知	00:25:b3:bf:91:ee

是什么使流量变得奇怪呢？回忆一下我们在第 6 章对 ARP 的讨论，ARP 数据包有两种类型：请求和响应。请求数据包在网络上广播给所有主机，用以发现包含特定 IP 地址的机器的 MAC 地址。接着，响应信息作为单播数据包发给请求的设备。在这个背景下，我们从通信序列中发现了一些奇怪的事情，参见图 12-16。

异步社区Kensuke(18319066421) 专享 请尊重版权

首先，数据包 54 是 MAC 地址为 00:25:b3:bf:91:ee 的攻击者发送的 ARP 请求，它作为单播数据包直接发送给 MAC 地址为 00:21:70:c0:56:f0 的受害者❶。这种类型的请求本应该广播给网络上所有主机，但它却只是直接发给了受害者。我们又注意到虽然这个数据包是攻击者发送的，并且在 ARP 头部包含了攻击者的 MAC 地址，但它却列出了路由器的 IP 地址，而不是它自己的。

紧随这个数据包的是受害者发给攻击者的响应，包含它的 MAC 地址信息❷。最诡异的事情出现在数据包 56 里：攻击者给受害者发送了一个包含未请求 ARP 响应的数据包，告诉它 172.16.0.1 对应的 MAC 地址是 00:25:b3:bf:91:ee❸。问题是 172.16.0.1 对应的 MAC 地址不是 00:25:b3:bf:91:ee 而应该是 00:26:0b:31:07:33。因为在之前的数据包捕获中我们看到过路由器 172.16.0.1 与受害者的通信，所以我们知道事实本应如此。由于 ARP 协议内在的不安全性（它的 ARP 表接收未请求的更新），因此现在受害者会将本应发送到路由器的流量发送给攻击者。

注意

因为这些数据包是从受害者机器上捕获的，所以你实际上没有看到事情的全貌。要使攻击生效，攻击者必须给路由器发送同样序列的数据包，骗它认为攻击者就是受害者。但我们需要在路由器（或攻击者）捕获才能看到这些数据包。

一旦两头都上当，受害者和路由器间的通信就会流经攻击者，如图 12-10 所示。

数据包 57 可以确认攻击取得成功。当你用神秘的 ARP 通信之前发送的数据包（比如数据包 40，参见图 12-11）与它比较时，就会发现远程服务器（Google）的 IP 地址是一样的❶，但目标 MAC 地址却变化了❷。MAC 地址的变化告诉我们，现在的流量抵达路由器之前将被路由到攻击者。

这个攻击如此狡猾，以至它很难被检测。要想发现它，你通常需要专门配置 IDS 的帮助，或者在设备上运行能检测 ARP 表项突然变化的软件。因为你很可能会想利用 ARP 缓存污染攻击来捕获网络上的数据包以便分析，所以了解如何使用这种技术也是很重要的。

异步社区Kensuke(18319066421) 专享 请尊重版权

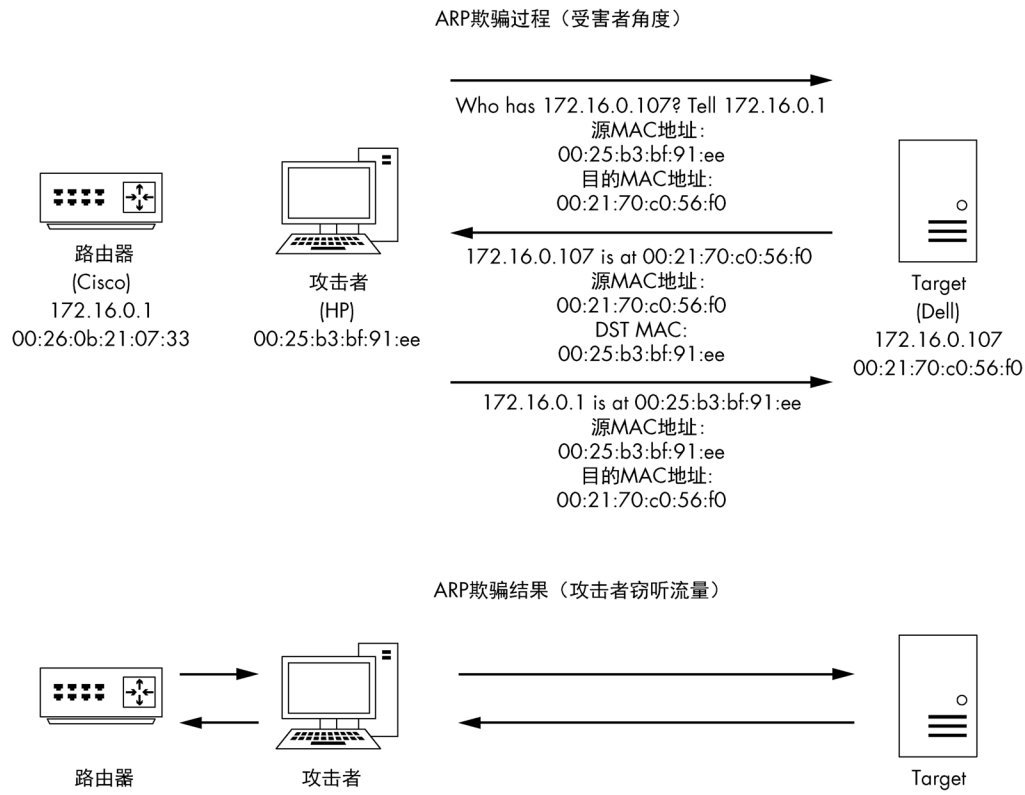


图 12-10　ARP 缓存污染导致 MITM 攻击

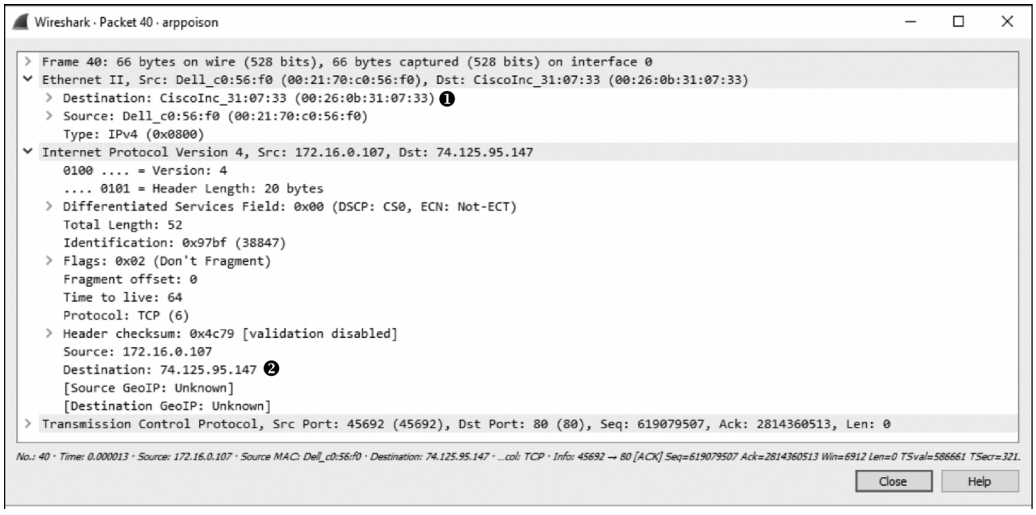


图 12-11　目标 MAC 地址的变化说明这次攻击是成功的

异步社区Kensuke(18319066421) 专享 请尊重版权

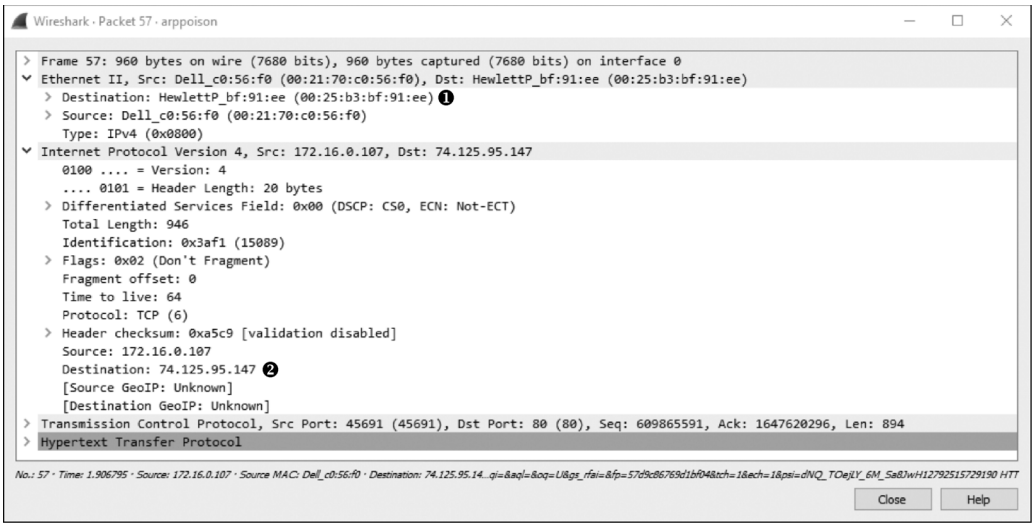


图 12-11　目标 MAC 地址的变化说明这次攻击是成功的（续）

12.2.2　会话劫持

现在你了解了如何恶意使用 ARP 缓存投毒。我现在将演示一种利用 ARP 缓存投毒的技术：会话劫持。在会话劫持中，攻击者窃取一个 HTTP 会话 cookie，然后伪装为另一个用户；我们将很快对 HTTP cookie 进行学习。为了达到这个目的，攻击者使用 ARP 缓存投毒截获一次目标通信，并找到相关的会话 cookie 信息。随后，攻击者能够使用窃取的 cookie，以对应用户的身份访问目标 Web 应用。

这个场景的通信数据在 sessionhijacking.pcapng 中。抓包文件包含目标（172.16.16.164）与 Web 应用（172.16.16.181）之间的传输数据包。用户在不知情的情况下成为攻击受害者，他们的通信过程被攻击者（172.16.16.154）主动监听。这些数据包在 Web 服务器上被抓取，这与会话劫持发生时，防御者的角度一致。

注意　本例中访问的 Web 应用是 Damn Vulnerable Web Application（DVWA）。此应用预留了很多可被多种攻击方式利用的漏洞，经常被作为教学工具使用。

抓取的通信过程基本由两个会话组成。第一个会话在目标用户和 Web 服务器之间进行，使用过滤器 `ip.addr == 172.16.16.164 && ip.addr ==`

异步社区Kensuke(18319066421) 专享 请尊重版权

172.16.16.181 分离此会话。这是一次正常的网页浏览通信，并没有特别之处。出于特殊的目的，我们主要关注请求中的 cookie 值。例如，在数据包 14 的 GET 请求中，你会在数据包详情窗口中发现 cookie，如图 12-12 所示。此处，cookie 使用 PHPSESSID 值 ncobrqrb7fj2a2sinddtk567q4❶标识会话 ID。

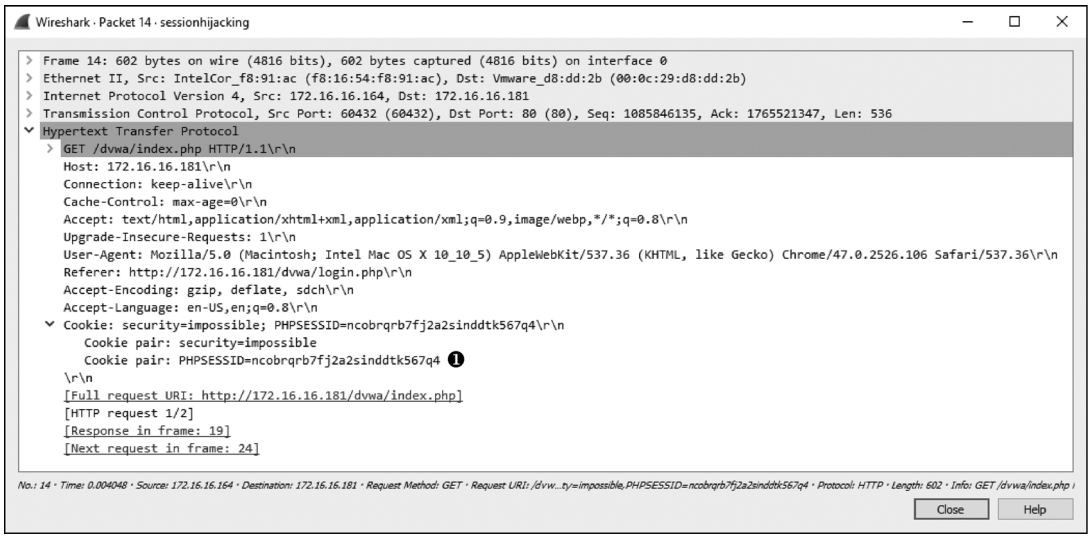


图 12-12　查看目标用户的会话 cookie

网站使用 cookie 维护每个用户的会话信息。当一个新用户访问网站时，用户将使用一个会话 ID 用于身份识别（本例中为 PHPSESSID）。在用户验证过程中，很多应用在用户使用会话 ID 完成验证后，在数据库中创建相应的记录，将会话 ID 作为已验证的会话凭证。任何使用这一 ID 的用户都能够使用此次验证记录接入应用。当然，开发者愿意相信，只有一个用户能够使用某一个特定的 ID，因为生成方式保证 ID 是独一无二的。然而，这种处理会话 ID 的方式是不安全的，因为恶意用户能够窃取其他用户的 ID，然后伪造身份。目前有一些方法可以用于防止会话劫持的发生，但是很多网站，包括 DVWA，仍然能够被会话劫持。

受害者没有意识到他们的通信正在被监听，或是发现他们的会话 cookie 被攻击者截取，如图 12-12 所示。现在，攻击者只需要使用窃取的 cookie 值即可与 Web 服务器通信。可以通过某些代理服务器完成这项工作，不过，使用浏览器插件会更简单，如 Chrome 的 Cookie 管理器。利用这个插件，攻击者可以将 PHPSESSID 设置为从以上通信获取的值，如图 12-13 所示。

异步社区Kensuke(18319066421) 专享 请尊重版权



图 12-13　使用 Cookie 管理器插件盗用受害者身份

如果你清除了之前设置的筛选条件，并浏览数据包，就会看到攻击者的 IP 地址与 Web 服务器的通信。你可以将筛选条件设为 `ip.addr == 172.16.16.154 && ip.addr ==172.16.16.181` 来限定查看范围。

在进一步研究之前，让我们添加一个栏目，用于在数据包列表面板中显示 cookie 值。如果在 ARP 缓存投毒的过程中添加了一些栏目，请先将它们删除。之后，按照在 ARP 缓存投毒章节提到的操作步骤新增栏目，栏目的字段名为 `http.cookie_pair`。添加完成后，将此栏目放在目的地址栏之后。界面看起来如图 12-14 所示。

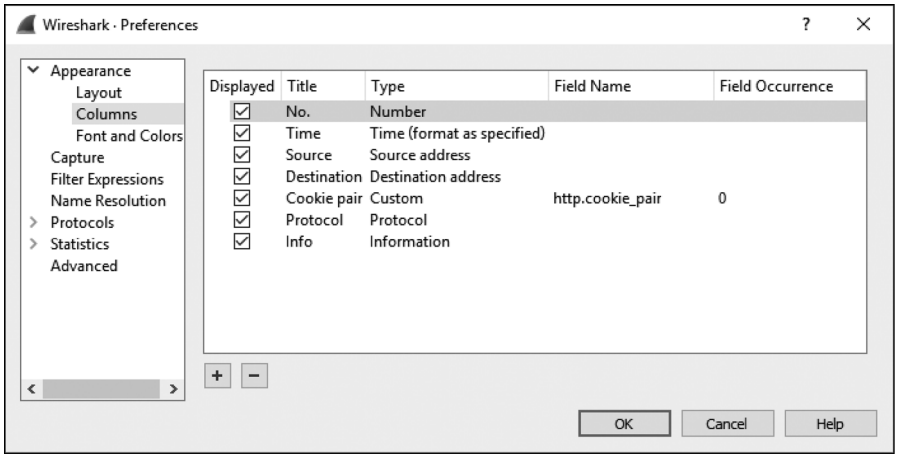


图 12-14　设置用于研究会话劫持的栏目

异步社区Kensuke(18319066421) 专享 请尊重版权

新的栏目设置完成后，修改筛选条件，仅显示 HTTP 请求，此处 TCP 通信没有用处。新的筛选条件为（ip.addr==172.16.16.154 && ip.addr==172.16.16.181) && (http.request.method || http.response.code)。过滤后的数据包如图 12-15 所示。

No.	Time	Source	Destination	Cookie pair	Protocol	Info
77	16.563004	172.16.16.154	172.16.16.181	security=low,PHPSESSID=lup70ajeuodkrhrvbmsjtgrd71	HTTP	❶GET /dvwa/ HTTP/1.1
79	16.565584	172.16.16.181	172.16.16.154		HTTP	HTTP/1.1 302 Found❷
80	16.570187	172.16.16.154	172.16.16.181	security=low,PHPSESSID=lup70ajeuodkrhrvbmsjtgrd71	HTTP	❸GET /dvwa/login.php HTTP/1.1
81	16.575123	172.16.16.181	172.16.16.154		HTTP	HTTP/1.1 200 OK (text/html)❹
115	60.040166	172.16.16.154	172.16.16.181	security=low,PHPSESSID=ncobrqrb7fj2a2sinddtk567q4	HTTP	❺GET /dvwa/ HTTP/1.1
118	60.042241	172.16.16.181	172.16.16.154		HTTP	HTTP/1.1 200 OK (text/html)❻
120	64.292056	172.16.16.154	172.16.16.181	security=low,PHPSESSID=ncobrqrb7fj2a2sinddtk567q4	HTTP	❼GET /dvwa/setup.php HTTP/1.1
122	64.293401	172.16.16.181	172.16.16.154		HTTP	HTTP/1.1 200 OK (text/html)❽

图 12-15　攻击者伪装为受害者

现在，我们查看攻击者和服务器之间的通信。在前 4 个数据包中，攻击者请求/dvwa/目录❶，接收到的响应状态码为 302；Web 服务器响应 302 表示请求重定向至其他 URL。此时，攻击者被重定向至登录页面/dvwa/login.php❷。攻击者的计算机请求登录页面❸，并返回为请求成功❹。两次请求均使用会话 ID lup70ajeuodkrhrvbmsjtgrd71。

随后，再次请求/dvwa/目录，我们注意到现在会话 ID❺发生了变化。会话 ID 现在是 ncobrqrb7fj2a2sinddtk567q4，与之前受害者使用的相同。这表明，攻击者操纵会话，使用了窃取的 ID。此时我们并没有被重定向至登录页面，请求返回了 HTTP 200 状态码，页面内容与登录后的受害者看到的一致❻。攻击者使用了受害者的 ID dvwa/setup.php❼，页面内容同样返回成功❽。攻击者和验证成功的受害者一样访问 DVWA 网站。这一过程中我们并不知道受害者的用户名或密码。

这只是攻击者将数据包分析变为攻击工具的一个例子。通常我们认为，当攻击者能够看到与通信过程相关的数据包时，将为恶意活动提供发生的可能性。这是安全专家提倡通过加密保护数据传输的原因之一。

12.3　漏洞利用

攻击者是吃饭还是喝粥，就要看漏洞利用阶段的表现了。攻击者已经研究并侦察好目标，并发现了准备利用的漏洞，期望以此进入目标系统。在本章剩下的篇幅里，我们将关注在一些漏洞利用技术中捕获的数据包。这些漏洞包括

异步社区Kensuke(18319066421) 专享 请尊重版权

近期的 Microsoft 漏洞、通过 ARP 缓存污染攻击重定向流量，以及盗窃数据的远程访问特洛伊木马。

12.3.1 极光行动

2010 年 1 月，极光行动利用了一个当时未知的 IE 浏览器漏洞。这个漏洞允许攻击者取得 Google 及其他公司的目标机器的 root 级别的远程控制权限。

用户只需要用包含该漏洞的 IE 浏览器访问一个网站，就能执行这个恶意代码。然后，攻击者就能立刻获得用户机器的管理员权限。攻击者使用了“网络钓鱼”引诱受害者。所谓“网络钓鱼”（Spear phishing）是指攻击者向受害者发送一封电子邮件，诱导受害者单击邮件中的链接，从而将其引导至一个恶意网站。通常，网络钓鱼信息看起来都像来自可信的源，因此它们经常能成功。

在极光行动的案例中，我们从用户单击钓鱼邮件中链接的那一刻开始讲起。这些数据包包含在文件 aurora.pcap 中。

这个捕获文件以受害者（192.168.100.206）和攻击者（192.168.100.202）之间的三次握手开始。初始化连接的目标是 80 端口，这使我们相信它是 HTTP 流量。第四个数据包证实了我们的假设，它是一个对/info 的 HTTP GET 请求❶，如图 12-16 所示。

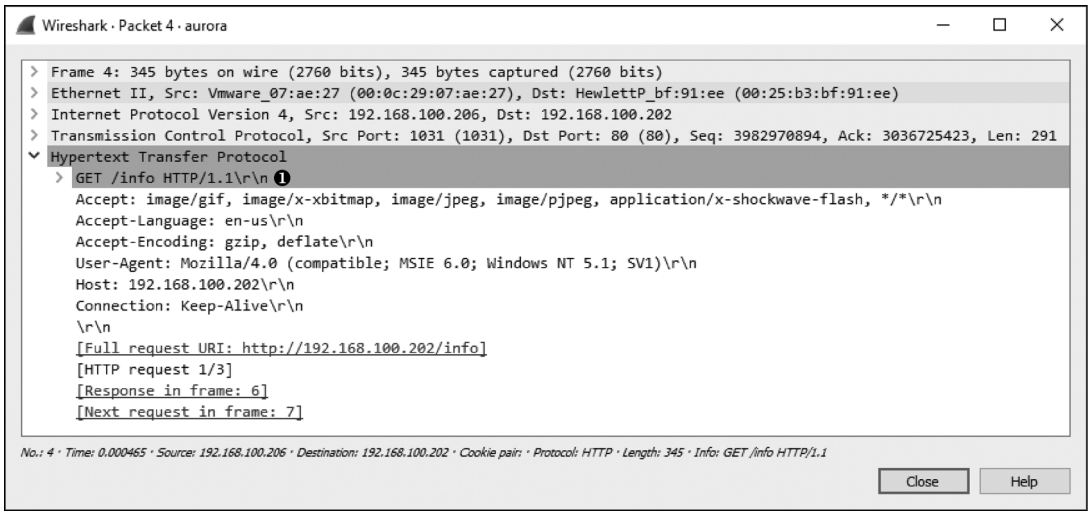


图 12-16　受害者做了一次针对/info 的 GET 请求

异步社区Kensuke(18319066421) 专享 请尊重版权

攻击者的机器确认收到了 GET 请求，并在数据包 6 中返回了一个 302（Moved Temporarily）码。这个状态码通常用于将浏览器重定向到另一个页面，在这个案例中正是如此。与 302 返回码❶一起来的还有一个 Location 字段，它指明重定向位置是/info?rFfWELUjLJHpP❷，如图 12-17 所示。

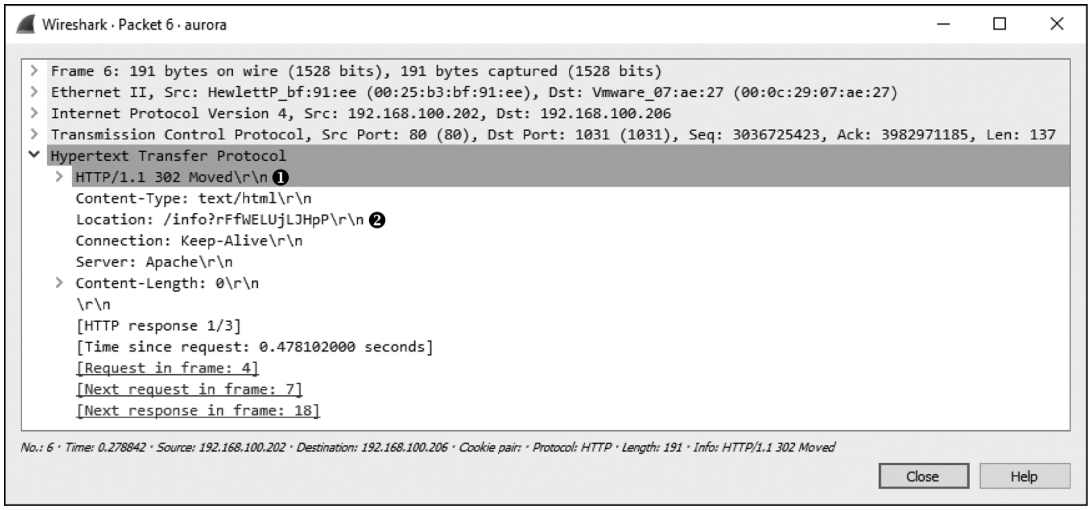


图 12-17　客户端浏览器被这个数据包重定向

收到 HTTP 302 数据包后，客户端在数据包 7 中初始化了另一个针对/info?rFfWELUjLJHpP 这个 URL 的 GET 请求，然后在数据包 8 中收到了一个 ACK。ACK 之后的几个包表示从攻击者发往受害者的数据。为了更好地查看那些数据，右击当前 TCP 流的任一个数据包，如数据包 9，选择 **Follow TCP Stream**。在这个数据流的输出中，我们看到了初始 GET 请求、302 重定向，以及第二个 GET 请求，如图 12-18 所示。

在此之后，事情变得十分奇怪了。攻击者向 GET 请求响应了一些看起来非常奇怪的内容。图 12-19 显示了这些内容的第一部分。

这些内容好像是<script>标签内的一系列随机数字和字母❶。HTML 内的<script>标签表示使用了一种高级脚本语言。在这个标签内，你通常会看到各种脚本语句。但这些乱码表明真正的内容可能已经被特殊编码以逃避检测了。由于我们知道这是一个漏洞利用的流量，因此我们可以假设这乱七八糟的文本包含了十六进制填充以及真正用于利用漏洞服务的 shellcode。

异步社区Kensuke(18319066421) 专享 请尊重版权

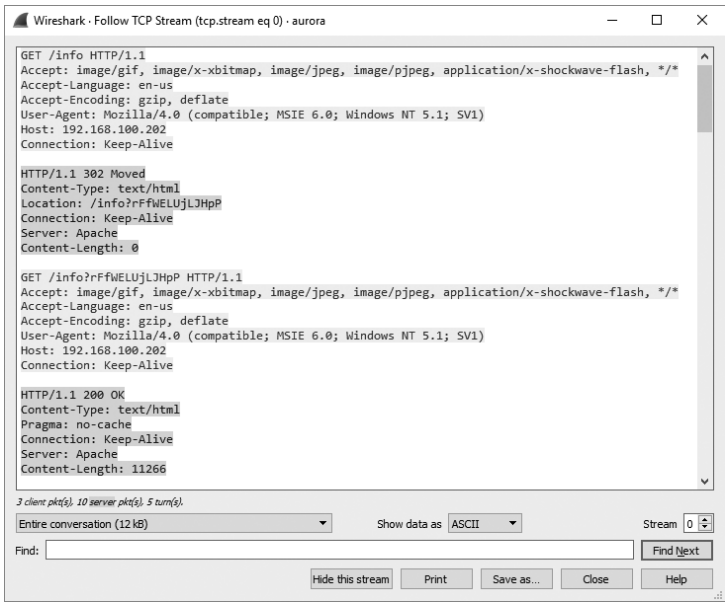


图 12-18　发送到客户端的数据流



图 12-19　<script>标签内的杂乱内容看起来像是被编码了

异步社区Kensuke(18319066421) 专享 请尊重版权

图 12-20 显示了攻击者发送的第二部分内容。在已编码文本之后，我们最终看到了一些可读文本。即便没有丰富的编程知识，我们也能看出这些文本像是在一些变量的基础上做一些字符串解析。这是闭合标签</script>之前的最后一些文本比特。

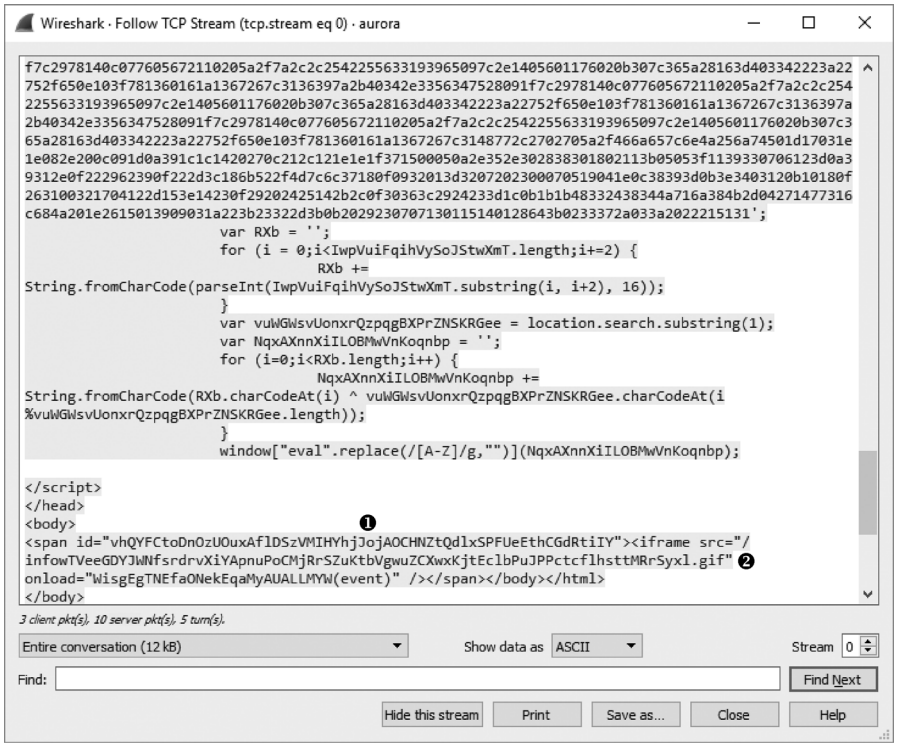


图 12-20 服务器发送的这部分内容包含了可读文本和可疑的 iframe

攻击者发给客户端的最后一段数据包含两个部分。第 1 部分是❶。第 2 部分包含在标签内，是<iframe src="/infowTVeeGDYJWNfsrdrvXiYApnuPoCMjRrSZuKtbVgwuZCXwxKjtEclbPuJPPctcflhsttMRrSyxl.gif"onload="WisgEgTNEfaONekEqaMyAUALLMYW(event)" />❷。这些内容也有可能是恶意行为的标志，因为这些文本出奇的长、包含不可读的随机字符串和可能被混淆过的文本。

标签内包含了一个 iframe，这是攻击者惯用的手法，用于在 HTML 页面中嵌入额外内容。<iframe>标签创建了一个内联帧，并不被用户察觉。在这

异步社区Kensuke(18319066421) 专享 请尊重版权

个案例中，<iframe>标签引用了一个名字古怪的 GIF 文件。如图 12-21 所示，当受害者的浏览器发现对这个文件的引用时，它在数据包 21 中发送了一个 GET 请求❶，紧接着 GIF 文件就被传送过来了❷。这个 GIF 文件有可能被用来以某种方式，触发已经下载到受害者机器上的漏洞利用代码。



No.	Time	Source	Destination	Protocol	Info
21	1.288241	192.168.100.206	192.168.100.202	HTTP	❶GET /infowTVeeGDYJWNfsrdrvXiYApnuPoCMjRrSZuKtbVgwuZCXwxKjtEclbPuJPPctcflhsttMRrSyxl.gif HTTP/1.1
22	1.488200	192.168.100.202	192.168.100.206	TCP	80 → 1031 [ACK] Seq=3036736951 Ack=3982971911 Win=64518 Len=0
23	1.489366	192.168.100.202	192.168.100.206	HTTP	❷HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)
24	1.650958	192.168.100.206	192.168.100.202	TCP	1031 → 80 [ACK] Seq=3982971911 Ack=3036737098 Win=64093 Len=0

图 12-21　受害者请求并下载 iframe 中指明的 GIF 文件

当受害者向攻击者的 4321 端口初始化连接时，文件中最奇怪的一部分出现了，请看数据包 25。从 Packet Details 面板中查看第二个通信流并不能得出太多信息，因此我们再次查看 TCP 通信流以更清楚地了解传输的数据。图 12-22 显示了 Follow TCP Stream 窗口的输出。

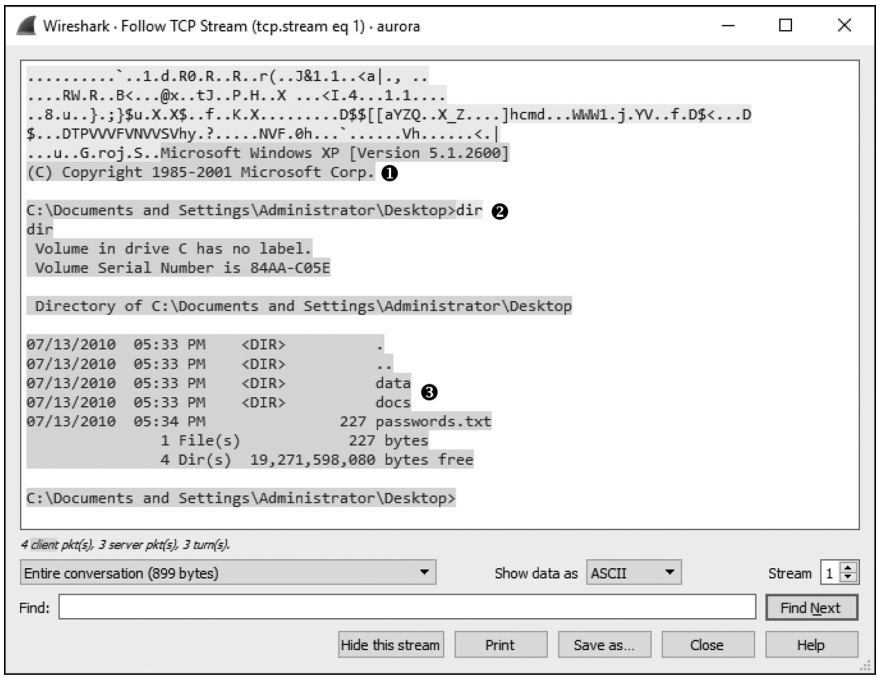


图 12-22　攻击者通过这个连接与命令窗口交互

在这里，我们看到了应该立即引发警报的东西：一个 Windows 的命令行解释器❶。这个 shell 是由受害者发给服务器的，这表明攻击者成功利用了漏洞：一旦漏洞利用程序启动，客户端就给攻击者发送回一个命令行解释器。在这个

异步社区Kensuke(18319066421) 专享 请尊重版权

捕获中，我们甚至能看到攻击者与受害者的交互：输入 dir 命令❷以列出受害者机器的目录内容❸。

攻击者进入这个命令行解释器后，对受害者机器就有了无限制的管理权限，他几乎能做任何想做的事情。受害者只不过是轻点了一下鼠标，几秒钟之内就把计算机的完全控制权限交给了攻击者。

像这样的漏洞利用程序在线路上传输信息时通常都编码成不可识别的字符，以避免被网络 IDS（入侵检测系统）发现。就其本身而言，没有对这个漏洞利用程序的事先了解，也没有该程序的代码样本，在没有进一步分析之前很难说清受害者系统上到底发生了什么。幸好，我们可以从数据包捕获中识别出恶意代码的一些蛛丝马迹。这包括<script>标签里的一些混淆文本、奇怪的 iframe，以及明文表示的命令行解释器。

我们在这里总结一下极光漏洞利用程序是如何工作的。

- 受害者收到一封来自攻击者的邮件，看起来合理，实际上有针对性，单击里面的一个链接，向攻击者的恶意网站发送一个 GET 请求。
- 攻击者的 Web 服务器向受害者发出 302 重定向，受害者的浏览器向重定向 URL 自动发起一个 GET 请求。
- 攻击者的 Web 服务器向客户端发送一个含有混淆的 JavaScript 代码（包括一个漏洞利用程序）的 Web 页面，以及一个含有恶意 GIF 图像链接的 iframe。
- 受害者向恶意图像发起一个 GET 请求，将它从服务器上下载下来。
- 利用 IE 浏览器的漏洞，使用恶意 GIF 文件解混淆之前发送的 JavaScript 代码，并在受害者机器上执行。
- 一旦漏洞被成功利用，就执行隐藏在混淆代码中的 payload，从受害者向攻击者的 4321 端口打开一个新会话。
- 该 payload 会产生一个命令行解释器并返回给攻击者，以便攻击者与目标进行交互。

从防御的观点看，我们可以使用这个捕获文件为 IDS 创建一个特征，也许能有助于检测这种攻击。举个例子，我们可以过滤捕获文件的非混淆部分，比如<script>标签里混淆文本末尾的明文代码。另外一个思路是为所有包含 302 重定向到特定 URL 的 HTTP 流量写一个特征。为能在生产环境中使用这个特征，还需要一些额外的调整，但这已经是个很好的开端。

异步社区Kensuke(18319066421) 专享 请尊重版权

注意 对尝试防御网络未知威胁的人而言，基于恶意流量样本创建流量特征是非常关键的一步。这里描述的捕获是提升编写特征技能的好方法。

12.3.2 远程访问特洛伊木马

到目前为止，我们已经利用先验知识查看了一些安全事件。这是学习攻击形态的好办法，但却不太符合实际。在真实世界里，守护网络安全的人不可能查看网络中的每一个数据包。他们会使用各式各样的 IDS 设备，基于预定义的攻击特征，来提醒他们留意网络流量里的异常情况，并实施进一步检查。

在下一个案例中，我们将像分析真正的网络威胁一样，从一个简单的警报开始。在这个例子中，我们的 IDS 生成了这个警报：

```
[**] [1:132456789:2] CyberEYE RAT Session Establishment [**]
[Classification:A Network Trojan was detected] [Priority:1]
07/18-12:45:04.656854 172.16.0.111:4433 -> 172.16.0.114:6641
TCP TTL:128 TOS:0x0 ID:6526 IpLen:20 DgmLen:54 DF
***AP*** Seq:0x53BAEB5E  Ack:0x18874922  Win:0xFAF0 TcpLen:20
```

下一步我们将查看触发此次警报的特征规则：

```
alert tcp any any -> $HOME_NET any (msg:"CyberEYE RAT Session Establishment";
content:"|41 4E 41 42 49 4C 47 49 7C|"; classtype:trojan-activity;
sid:132456789; rev:2;)
```

这个规则是这样定义的：当它发现一个进入内网的数据包含有十六进制内容 41 4E 41 42 49 4C 47 49 7C 时，就产生警报。这个内容转换成可读 ASCII 码是 ANA BILGI。当检测到这个字符串响起警报时，可能预示着 CyberEYE 远程访问木马（Remote-access Trojan，RAT）的出现。RAT 是秘密运行在受害者计算机上的恶意应用程序，它向攻击者建立连接，使攻击者能远程管理受害者的机器。

注意 CyberEYE 是来自土耳其的工具，用于产生 RAT 程序和管理“肉鸡”。有趣的是，这里看到的 Snort 规则有一个敏感字符串“ANA BILGI”，它其实是土耳其文字，表示“基本信息”的意思。

现在我们将在 ratinfected.pcap 文件中查看与警报有关的流量。Snort 通常只捕获触发警报的单个数据包，但幸好我们有主机间的完整通信序列。让我们搜索 Snort 规则中提到的十六进制字符串，直接跳到关键部分。

异步社区Kensuke(18319066421) 专享 请尊重版权

（1）选择 **Edit->Find Packet**。

（2）选择 **Hex Value** 单选按钮。

（3）在文本框输入 **41 4E 41 42 49 4C 47 49 7C**。

（4）单击 **Find**。

如图 12-23 所示，你最先在数据包 4 的数据部分发现了以上字符串❶。

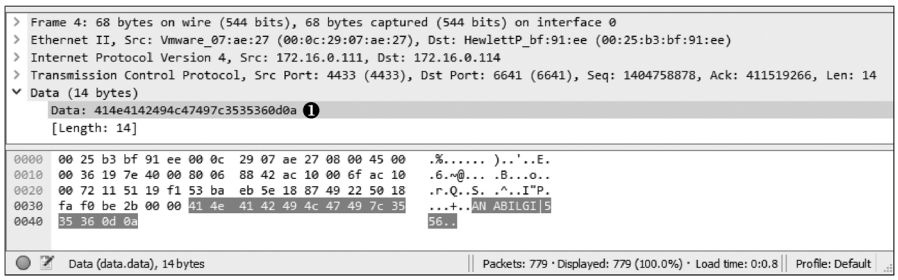


图 12-23　最先在数据包 4 中发现了 Snort 警报中的字符串内容

若多次选择 **Edit->Find Next**，你会看见这个字符串也在数据包 5、10、32、156、280、405、531 和 652 出现了。虽然这个捕获文件里的所有通信都是在攻击者（172.16.0.111）和受害者（172.16.0.114）之间产生的，但看起来这个字符串出现在了不同的会话中。数据包 4 和 5 使用 4433 端口和 6641 端口通信，而其他大部分实例出现在 4433 端口和其他随机选择的临时端口之间。通过查看 Conversations 窗口的 TCP 标签，我们可以确认存在多个会话，如图 12-24 所示。

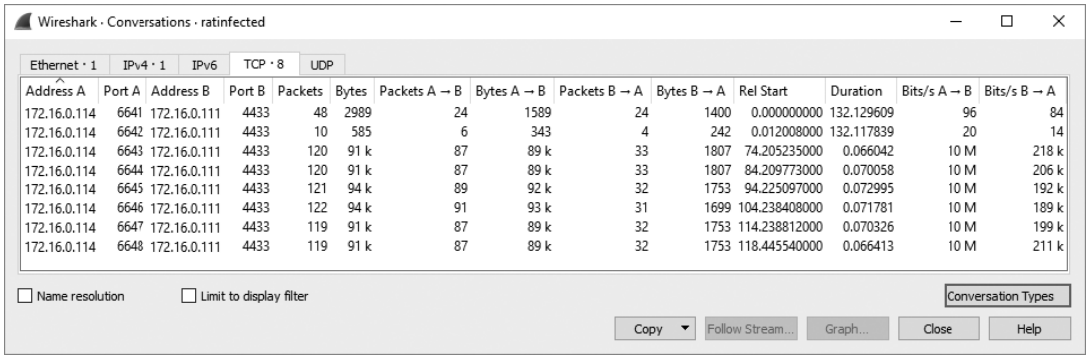


图 12-24　攻击者和受害者之间存在 3 个独立的会话

异步社区Kensuke(18319066421) 专享 请尊重版权

我们可以给捕获文件里的不同会话刷上不同颜色，将他们从视觉上分开。

（1）在 Packet List 面板上面的过滤器栏里，输入过滤器`(tcp.flags.syn == 1) && (tcp.flags.ack == 0)`。然后单击 **Apply**。这将筛选出流量中每一个会话的初始 SYN 数据包。

（2）右击第一个数据包，选择 **Colorize Conversation**。

（3）选择 **TCP**，然后选择一种颜色。

（4）为剩下的 SYN 数据包重复这个过程，分别选择不同的颜色。

（5）完成之后，选择 **Clear** 移除过滤器。

为每个会话着色后，我们可以看一看他们之间有什么关联，这将帮助我们更好地跟踪两台主机之间的通信过程。第一个会话（ports 6641/4433）是两端通信的开端，从这开始不会错了。右击会话里的任一个数据包，选择 **Follow TCP Stream** 可以看到被传输的数据，如图 12-25 所示。

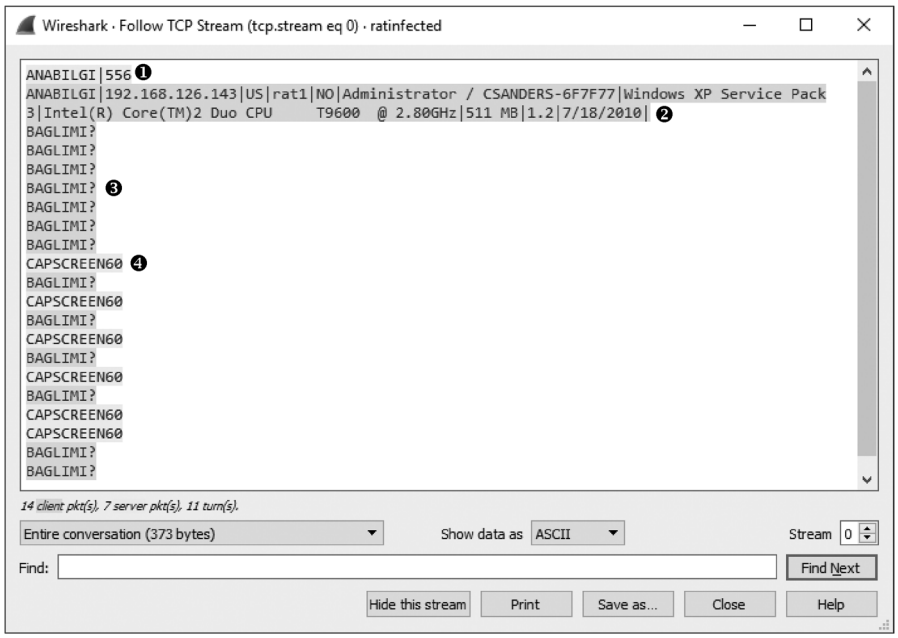


图 12-25　第一个会话产生了有趣的结果

很快，我们看到攻击者给受害者发送文本字符串“ANABILGI|556”❶。结果，受害者响应了一些基本系统信息，包括计算机名称（CSANDERS-6F7F77）和

异步社区Kensuke(18319066421) 专享 请尊重版权

使用的操作系统（Windows XP Service Pack 3）❷，并开始给攻击者发送回一些重复的字符串“BAGLIMI?”❸。攻击者返回的消息只有字符串“CAPSCREEN60”❹，出现了 6 次。

攻击者返回的字符串“CAPSCREEN60”很有意思，看一看它要带我们去哪里。我们再次使用 search 对话框在数据包中搜索这个文本字符串，指明 **String** 选项。

我们搜索到数据包 27 首次出现了这个字符串。这个信息的有趣之处在于，客户端一收到攻击者发送的这个字符串，就确认接收这个数据包，并在数据包 29 发起了一个新会话。

现在，如果我们跟随这个新会话（见图 12-26）的 TCP 流输出，就能看到熟悉的字符串“ANABILGI|12”，跟在后面的是字符串“SH|556”，最后是字符串“CAPSCREEN|C:\WINDOWS\jpgevhook.dat|84972”❶。注意到字符串“CAPSCREEN”之后指明的文件路径后面跟着不可读的文本。这里最吸引人的是不可读文本有一个前导字符串“JFIF”❷，Google 搜索一下知道，JPG 文件的开头部分通常就包含它。

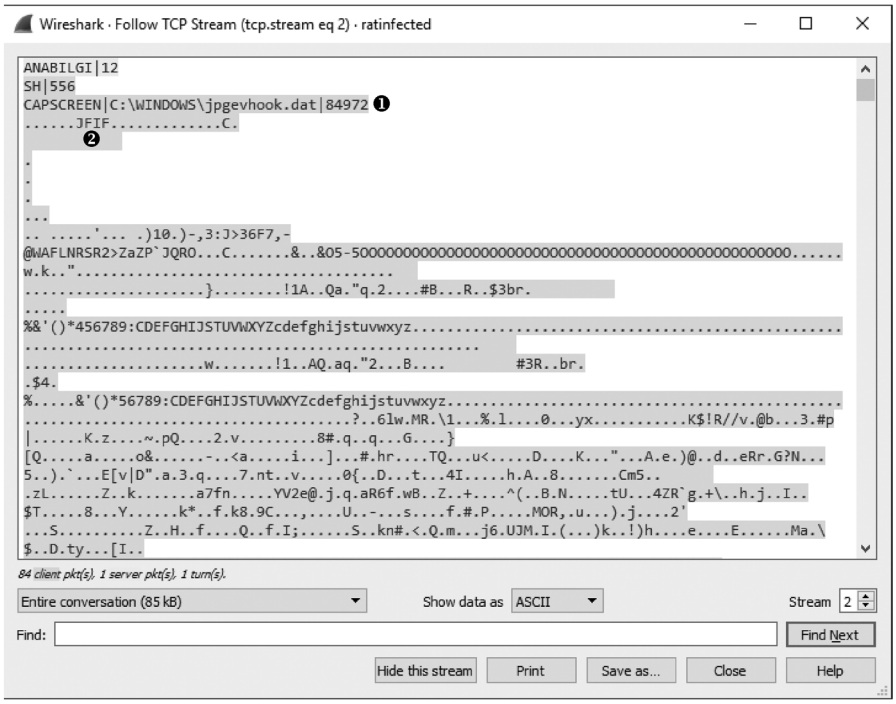


图 12-26　似乎攻击者发起了对一个 JPG 文件的请求

异步社区Kensuke(18319066421) 专享 请尊重版权

到这里，我们可以负责任地说，攻击者发起新会话是为了传输这个 JPG 图像。但更重要的是，我们开始从流量中推断出一个命令结构。看起来，攻击者发送“CAPSCREEN”命令就可以发起 JPG 图像的传输。实际上，不管什么时候发送“CAPSCREEN”命令，结果都是一样的。为了验证这个结论，可以查看每个会话的流，或使用下面介绍的 Wireshark 的 IO 绘图功能。

（1）选择 **Statistics->IO Graphs**。

（2）在 5 个过滤器栏中分别插入 `tcp.stream eq 2`、`tcp.stream eq 3`、`tcp.stream eq 4`、`tcp.stream eq 5` 和 `tcp.stream eq 6`。

（3）单击 **Graph 1**、**Graph 2**、**Graph 3**、**Graph 4** 和 **Graph 5** 按钮分别启用各个过滤器的数据点。

（4）将 y 轴的单位改成 **Bytes/Tick**。

图 12-27 显示了结果图像。

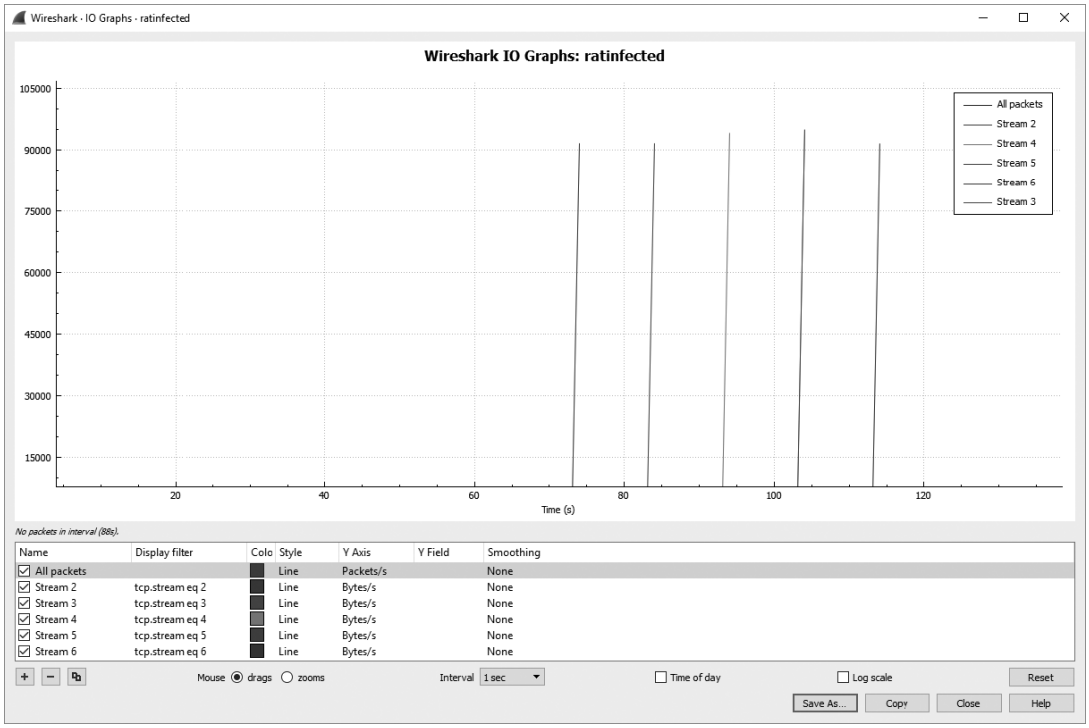


图 12-27　图像显示同样的活动重复了多次

从图 12-27 来看，似乎每个会话都包含同样大小的数据，并出现了同样多的

异步社区Kensuke(18319066421) 专享 请尊重版权

时间。现在我们可以总结，这个活动重复了好几次。

对于传输的 JPG 图片内容，你可能已经有了些想法，所以让我们看一看是否能查看这些 JPG 文件。执行以下步骤可以从 Wireshark 中提取 JPG 数据。

（1）首先，对特定数据包的 TCP 流进行重组，如之前图 12-22 所示的文本。

（2）然后通信被分离出来，我们只能看到受害者发送给攻击者的流数据。选择下拉菜单旁边的箭头，那里写着 `Entire Conversation (85033 bytes)`。确保选择合适的流量方向，就是 `172.16.0.114:6643 --> 172.16.0.111:4433 (85020 bytes)`。

（3）选择 **Save As** 按钮保存数据，确保扩展名是.jpg。

你现在试一下，一定会发现它无法打开。因为我们还需要再做一步。不像在第 8 章从 FTP 流量中提取完整文件那样，这里的流量在真实数据之外还增加了一些额外内容。在这个例子中，TCP 流的前两行实际上是木马命令序列的一部分，而不属于 JPG 数据（见图 12-28）。当我们保存数据流时，这些额外的数据也被保存了下来。结果，文件浏览器查找 JPG 文件头时遇到了预料之外的信息，导致它不能打开图片。

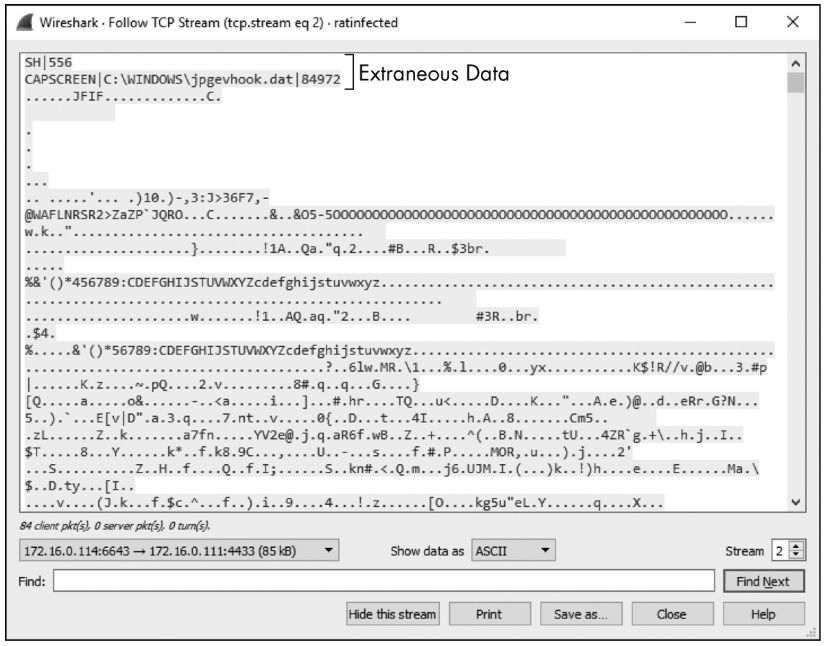


图 12-28　特洛伊木马增加的额外数据导致文件无法正确打开

异步社区Kensuke(18319066421) 专享 请尊重版权

用十六进制编辑器修复这个问题很简单。这个过程叫“文件修复”（File carving）。在图 12-29 中，我已经用 WinHex 选定 JPG 文件前面的一些字节。你可以使用任何十六进制编辑器删除这些字节并保存图片文件。

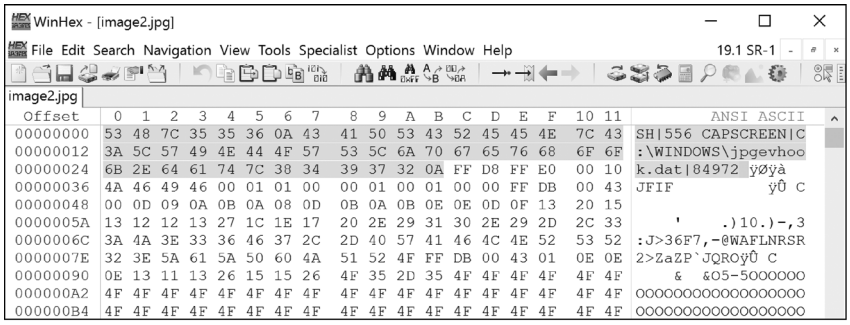


图 12-29　移除 JPG 文件中的附加字节

移除不需要的数据后，文件应该能够打开了。很显然，木马将受害者的桌面进行截屏，并发送回给攻击者（见图 12-30）。



图 12-30　发送的 JPG 文件是受害者计算机的屏幕截图

异步社区Kensuke(18319066421) 专享 请尊重版权

这些通信序列完成之后，通信就随着 TCP 拆除包而结束了。

这个场景展示的思考过程，就是一位入侵分析师分析 IDS 警报流量时，应遵循的典型步骤。

（1）查看警报及触发它的特征。

（2）在恰当的上下文确定特征确实在流量中。

（3）查看流量，找出攻击者对“肉鸡”动了什么手脚。

（4）在“肉鸡”泄露更多敏感信息之前，控制局面。

12.4 漏洞利用工具包和勒索软件

在最后一个场景中，我们将研究 IDS 的一次告警。我们将查看从被感染的系统中采集到的实时数据包，并尝试对攻击行为进行溯源。我们将在本例中使用可能会从身边设备中发现的真实恶意软件。

Snort 的 Sguil 控制台进行了一次 IDS 告警，如图 12-31 所示。Sguil 是用于对一个或多个感应器发出的告警进行管理、查看和研究的工具。它提供一个几乎是最友好的用户界面，是在安全研究人员中很流行的工具。

在 Sguil 中有很多关于这次告警的信息。顶部的窗口❶显示了告警的概要信息。包括在告警发生时的源和目的 IP 地址、端口，协议，根据匹配的 IDS 标识产生的事件消息。在本例中，本地系统 192.168.122.145 正在与位于 184.170.149.44 的未知外部系统进行通信，外部系统使用的端口为 80，此端口通常与 HTTP 通信联系起来。这个外部系统被认为是恶意主机，告警显示，该系统与一个识别恶意通信的标识有关，并且关于此系统，我们了解的信息很少。这次通信匹配到的标识代表了由 CyptoWall 恶意软件族发起的签到流量，这意味着一个该恶意软件的变体被安装在内部系统中。

Sguil 控制台提供了匹配规则的语法❷和单个数据包中与规则相匹配的数据❸。我们注意到，数据信息被拆分为协议头和数据部分，与 Wireshark 中的数据包信息展现形式类似。不幸的是，Sguil 仅提供单个被匹配的数据包的信息，但是我们需要更深入的研究。下一步，在 Wireshark 中检查与这次告警相关的通信，尝试定位告警流量，并查看出现的问题。这些流量在 cryptowall4_c2.pcapng 中。

异步社区Kensuke(18319066421) 专享 请尊重版权

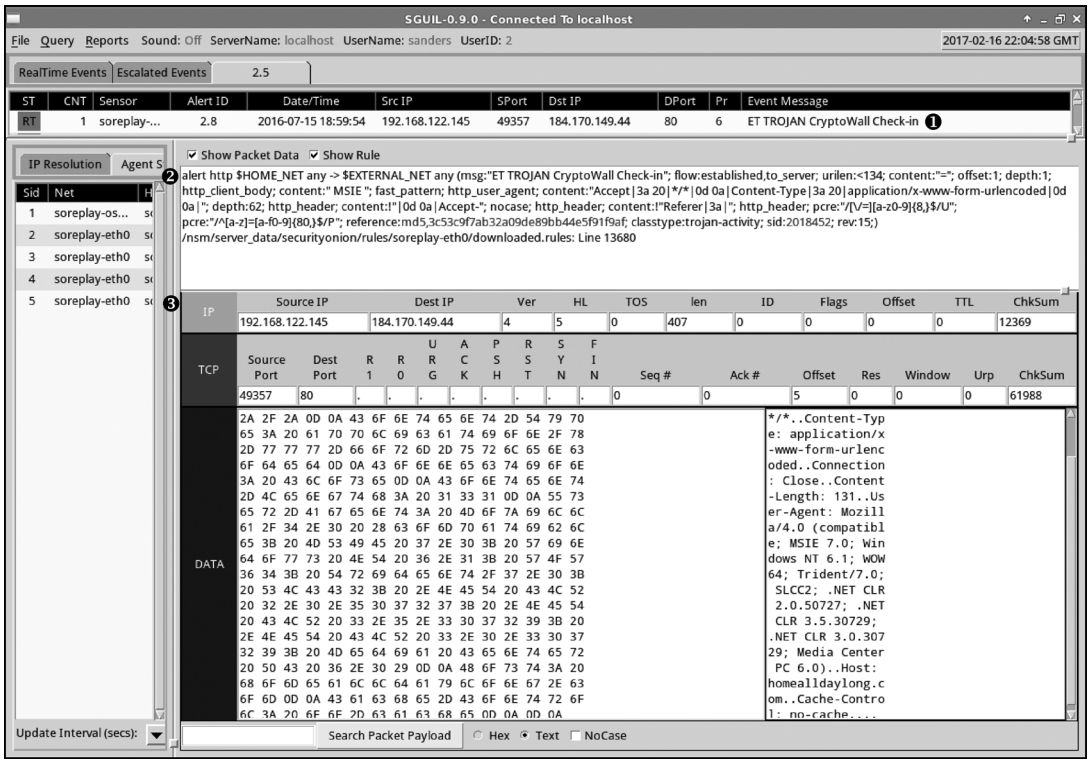


图 12-31　IDS 告警表明一次 CryptoWall 4 感染

这个抓包文件包含了告警时间前后的通信过程，数据并不是太复杂。第一次会话出现在数据包 1～16 中，我们可以跟踪会话的 TCP 流（见图 12-32）以便于查看。在抓包文件的开头，本地系统建立了到恶意主机 80 端口的 TCP 连接，之后向一个 URL ❶发起了一次 POST 请求，请求包含少量带有文字与数字的数据❷。在连接正常断开前，恶意主机回复了一个由文字和数字组成的字符串❹和 HTTP 200 OK 响应码。

如果浏览抓包文件的剩余部分，你会发现以上的序列在这些服务器之间重复出现，每次传输的数据量有差异。设置筛选条件为 `http.request.method == "POST"`，看到出现了 3 次类似 URL 结构的连接（见图 12-33）。

请求页面部分（76N1Lm.php）保持一致，但是余下的内容（被发至该页面的参数和数据）有变动。这个重复的通信序列和请求结构，与恶意软件的命令和控制（C2）行为一致，并符合此 IDS 告警标识的匹配规则。你可以在流行的 Cyrpto 研究网站 Crypto Tracker 上看到一个类似的例子，从而进一步核实。

异步社区Kensuke(18319066421) 专享 请尊重版权

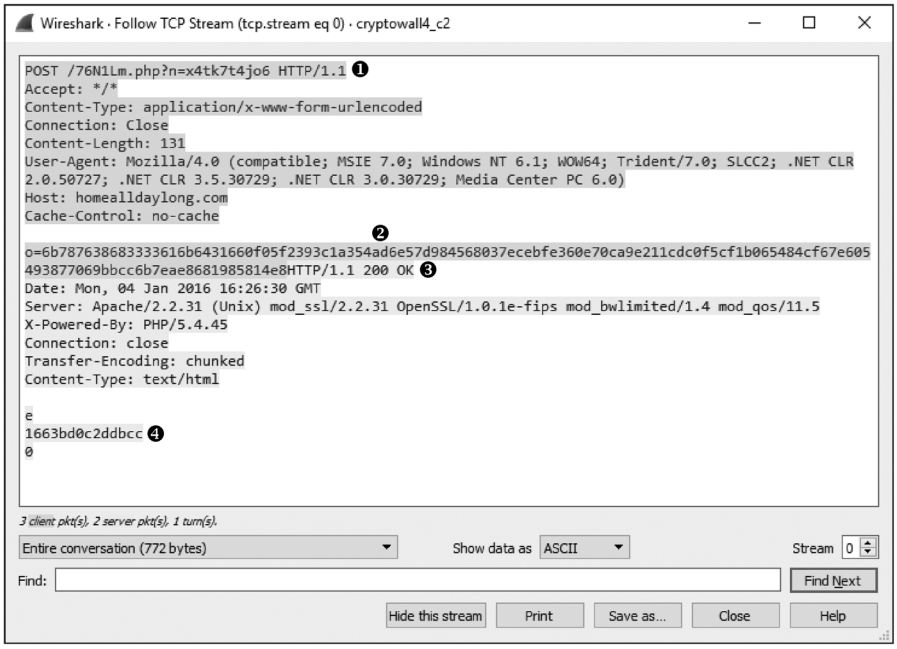


图 12-32　少量信息通过 HTTP 在服务器之间传输

No.	Time	Source	Destination	Protocol	Info
6	0.491136	192.168.122.145	184.170.149.44	HTTP	POST /76N1Lm.php?n=x4tk7t4jo6 HTTP/1.1 (application/x-www-form-urlencoded)
22	15.545562	192.168.122.145	184.170.149.44	HTTP	POST /76N1Lm.php?g=9m822y31lxud7aj HTTP/1.1 (application/x-www-form-urlencoded)
152	41.886948	192.168.122.145	184.170.149.44	HTTP	POST /76N1Lm.php?i=ttfkjb668o38k1z HTTP/1.1 (application/x-www-form-urlencoded)

图 12-33　URL 结构显示，不同的数据被发送至同一个页面

既然已经确定发生了基于恶意软件的 C2 通信，我们就应该采取补救措施，并定位被感染的机器。当问题涉及例如 CryptoLocker 的恶意软件时，这样的措施尤为重要，因为这类恶意软件会试图加密用户数据，除非用户支付高额赎金，否则攻击者不会提供解密密钥——这类恶意软件被称为勒索软件。被勒索软件感染的补救措施不在本书的讨论范围内；在现实场景中，这是安全分析人员的后续研究内容。

通常，下一个需要考虑的问题是，内部系统最初是如何被恶意软件感染的。如果答案能够被确定，那么你可能会发现其他设备因为类似的原因被另外的恶意软件感染；或者能够开发防护工具、制定检测机制，避免可能发生的感染。

告警数据包仅显示了被感染后的 C2 序列。在采用了安全监控和持续抓包的网络中，很多感应器被设置为存储几小时或几天内的流量数据，用于取证分析。毕竟，不是每一个机构都使用了能够实时告警的设备。数据包的临时存储让我

异步社区Kensuke(18319066421) 专享 请尊重版权

们能够查看，在前面看到的 C2 序列开始之前，被感染主机发出的数据。这些流量数据在 ek_to_cryptowall4.pcapng 中。

在这个抓包文件中有更多的数据包，新增数据包都是 HTTP 通信数据。我们已经了解了 HTTP 是如何工作的，现在让我们切入正题，设置过滤条件为 http.request，仅查看 HTTP 请求。筛选结果显示了 11 次由内部主机发起的 HTTP 请求（见图 12-34）。



No.	Time	Source	Destination	Protocol	Info
4	0.534405	192.168.122.145	113.20.11.49	HTTP	GET /index.php/services HTTP/1.1
35	5.265859	192.168.122.145	45.32.238.202	HTTP	GET /contrary/1653873/quite-someone-visitor-nonsense-tonight-sweet-await-gigantic-dance-third HTTP/1.1
39	6.109508	192.168.122.145	45.32.238.202	HTTP	GET /occasional/bXJkeHFlYXhmaA HTTP/1.1
123	9.126714	192.168.122.145	45.32.238.202	HTTP	GET /goodness/1854996/earnest-fantastic-thorough-weave-grotesque-forth-awaken-fountain HTTP/1.1
130	14.020289	192.168.122.145	45.32.238.202	HTTP	GET /observation/enVjZ2dtcnpz HTTP/1.1
441	30.245463	192.168.122.145	213.186.33.18	HTTP	POST /VOEHSQ.php?v=x4tk7t4jo6 HTTP/1.1 (application/x-www-form-urlencoded)
456	41.772768	192.168.122.145	184.170.149.44	HTTP	POST /76N1Lm.php?n=x4tk7t4jo6 HTTP/1.1 (application/x-www-form-urlencoded)
472	45.628284	192.168.122.145	213.186.33.18	HTTP	POST /VOEHSQ.php?w=9m822y311xud7aj HTTP/1.1 (application/x-www-form-urlencoded)
487	56.827194	192.168.122.145	184.170.149.44	HTTP	POST /76N1Lm.php?g=9m822y311xud7aj HTTP/1.1 (application/x-www-form-urlencoded)
619	71.971402	192.168.122.145	213.186.33.18	HTTP	POST /VOEHSQ.php?h=ttfkjb668o38k1z HTTP/1.1 (application/x-www-form-urlencoded)
634	83.168580	192.168.122.145	184.170.149.44	HTTP	POST /76N1Lm.php?i=ttfkjb668o38k1z HTTP/1.1 (application/x-www-form-urlencoded)

图 12-34　11 个由内部主机发起的 HTTP 请求

第一个请求由内部主机 192.168.122.145 发往未知外部主机 113.20.11.49。查看数据包的 HTTP 部分（见图 12-35），我们发现用户请求了一个页面❶，页面入口为 Bing 搜索这个网址的结果❷。到目前为止，看起来一切正常。



图 12-35　一次发往未知外部主机的 HTTP 请求

接下来，内部主机在数据包 35、39、123 和 130 中发送了 4 个向另一个未知外部主机 45.32.238.202 的请求。在之前章节的例子中我们了解到，当浏览器访问的页面包含外部内容引用或第三方服务器上的广告时，从网站服务器以外的主机获取内容是很常见的。虽然请求的 URL 看起来有些杂乱，但是这个请求本身并不令人担忧。

异步社区Kensuke(18319066421) 专享 请尊重版权

从数据包 39 中的 GET 请求开始，数据开始值得注意。跟踪这次交互的 TCP 流（见图 12-36），你会发现内部主机请求了名为 bXJkeHFlYXhmaA 的文件❶。这个文件名比较奇怪，并且没有文件扩展名。

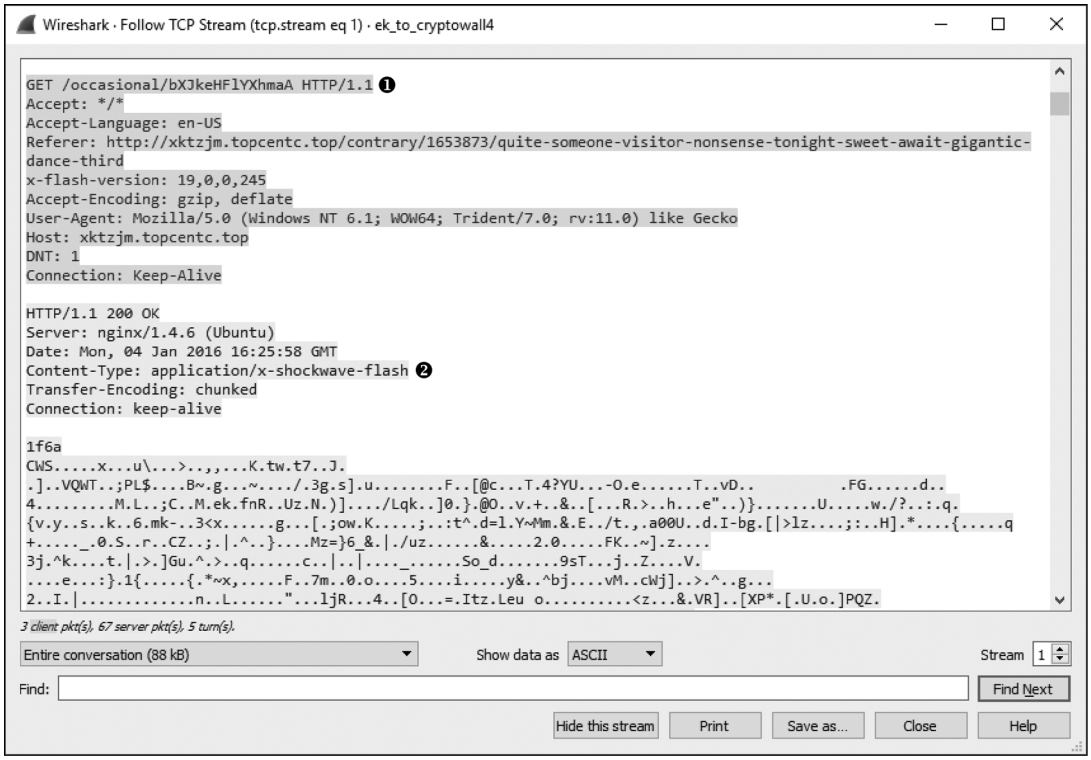


图 12-36　一个名称奇怪的 Flash 文件被下载

在更细致的检查中，我们看到 Web 服务器将文件内容识别为 x-shockwave-flash❷类型。Flash 是浏览器上流行的流媒体插件。设备下载 Flash 内容很正常，但是值得注意的是，Flash 由于有很多软件漏洞而臭名昭著，并且，Flash 经常不修补漏洞。随后，Flash 文件被成功下载。

Flash 文件被下载后，在数据包 130 中有一个类似的请求——请求一个名称奇怪的文件。跟踪这个 TCP 流（见图 12-37），你会看到请求的文件名为 enVjZ2dtcnpz❶。这个文件的类型没有被扩展名标明或被服务器识别。这个请求完成后，客户端下载了一个大小为 358400 字节的无法阅读的数据文件❷。

该文件下载完毕后不到 20s，你会看到在图 12-34 中出现过的一系列 HTTP 请求。从数据包 441 开始，内部服务器向两个不同的服务器发送相同的 C2 格式

异步社区Kensuke(18319066421) 专享 请尊重版权

HTTP POST 请求。我们可能发现了感染的源头。下载的两个文件是问题的原因。数据包 39 中请求的第一个文件是一个 Flash 漏洞利用程序，在数据包 130 中请求的第二个文件是恶意软件。

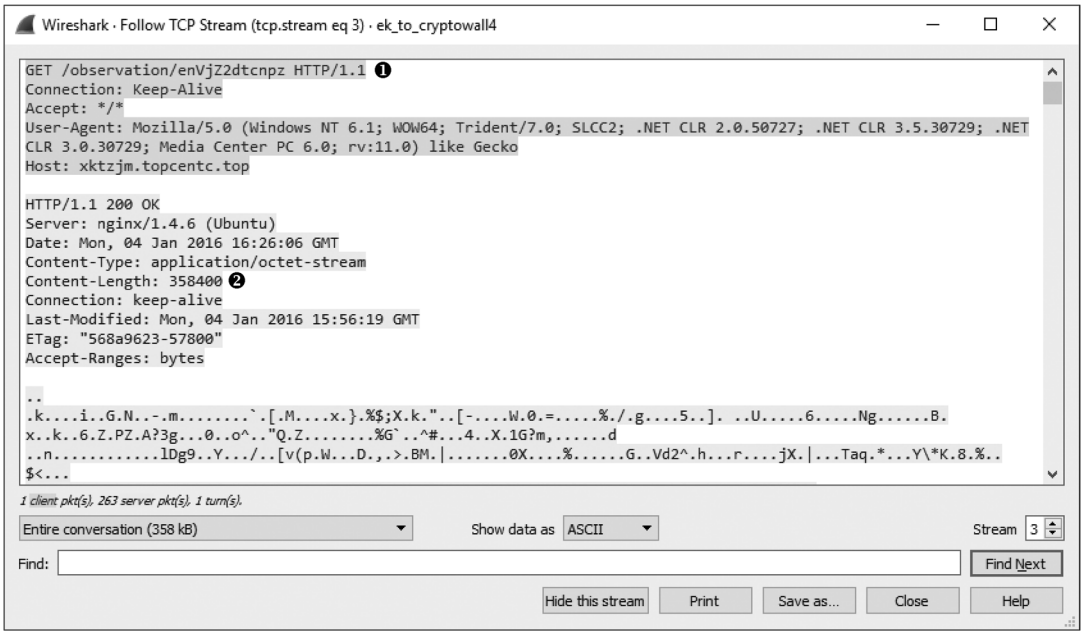


图 12-37　另一个名称奇怪的文件被下载，其文件名未被识别

注意　你可以使用恶意软件分析技术，对流量数据中包含的文件进行解码和分析。如果你对恶意软件逆向工程感兴趣，我推荐 MIchael Sikorski 和 Andrew Honig 的《恶意代码分析实战》(2012)，另一本 No Starch Press 的书，也是我个人最喜欢的书之一。

这个场景代表了最常见的恶意软件感染技术之一。用户在浏览网页时，误入了一个被漏洞利用工具包插入了恶意重定向代码的网站。这些漏洞利用程序感染正规网站，并具备采集客户端指纹信息的功能，以确定客户端存在的漏洞。被感染的页面被称为漏洞利用工具着陆页，它的目的是，根据工具包确定的用户系统存在的漏洞，将用户客户端重定向至另一个含有对应漏洞利用程序的站点。

你刚才看到的数据包来源于 Angler 漏洞利用工具包，其可能是 2015 年和 2016 年最常见的工具包。当用户访问一个被 Angler 感染的网站时，此工具包会辨别用户系统是否存在一个特定的 Flash 漏洞。随后，用户下载一个 Flash 软件，

异步社区Kensuke(18319066421) 专享 请尊重版权

用户系统被感染，用户下载第二个载荷——CryptoWall 恶意软件并安装。整个过程如图 12-38 所示。

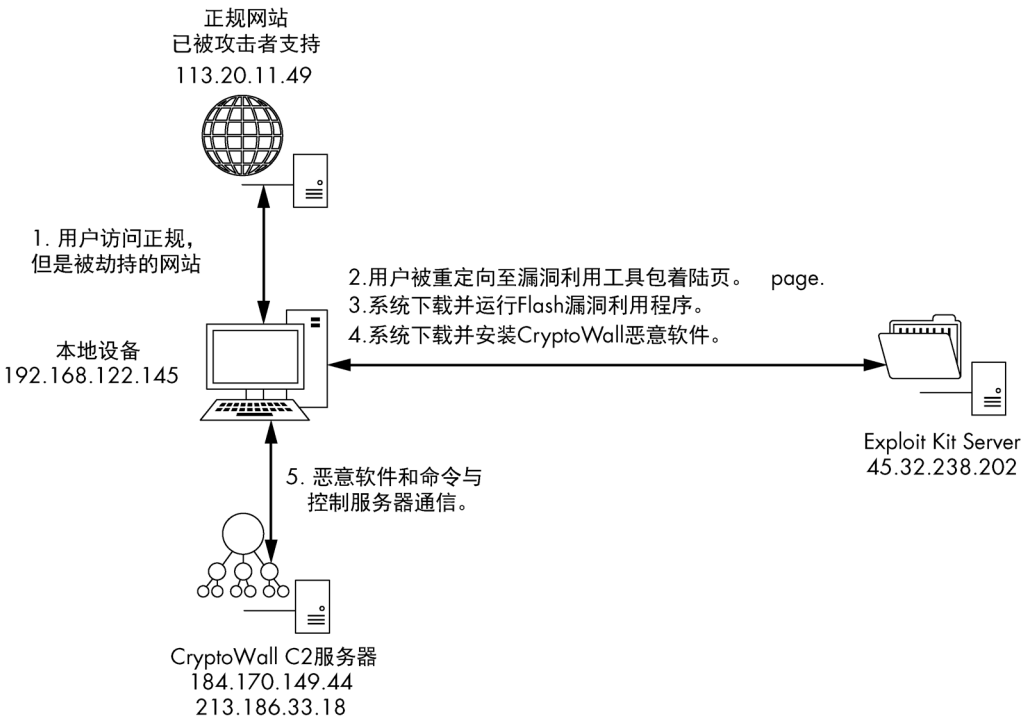


图 12-38 漏洞利用工具包感染过程

12.5 小结

在本章，我们讲解了如何在安全相关的场景中解析数据包捕获、分析常见攻击技术以及响应 IDS 警报等内容。这些内容足可以写一本书。而我们只是学习了一些常用的扫描和枚举类型、一种常见的中间人攻击技术、两个如何利用系统漏洞的例子，以及一旦发生这些情况会有什么后果而已。

异步社区Kensuke(18319066421) 专享 请尊重版权

第13章

无线网络数据包分析



与传统有线网络相比，无线网络稍微有些不同。虽然我们仍然在处理 TCP、IP 等常见的通信协议，但是当移到 OSI 模型最底层时，游戏便发生了一点变化。在这里，由于无线网络和物理层的本质属性，数据链路层变得尤为重要。这给我们捕获和访问数据增加了新的限制。

考虑到这些额外因素，你应该不会惊讶于这一整章都将讨论无线网络中的数据包捕获和分析。本章我们将讨论为什么无线网络在数据包分析中比较特殊，以及如何克服这些困难。当然，我们会通过捕获无线网络的实际例子来进行说明。

13.1 物理因素

在无线网络中捕获和分析传输数据，首先考虑的是物理传输介质。到目前

异步社区Kensuke(18319066421) 专享 请尊重版权

为止，我们都没有考虑物理层，因为我们一直在物理的线缆上通信。现在我们通过不可见的无线电波通信，数据包就从我们身边飞过。

13.1.1 一次嗅探一个信道

当从无线局域网（Wireless Local Area Network，WLAN）捕获流量时，最特殊的莫过于无线频谱是共享介质。不像有线网络的每个客户端都有它自己的网线连接到交换机，无线通信的介质是客户端共享的空域。单个 WLAN 只占用 802.11 频谱的一部分。这允许同一个物理空间的多个系统在频谱不同的部分进行操作。

注意 无线网络的基础是美国电子和电气工程师协会（Institute of Electrical and Electronics Engineers，IEEE）开发的 802.11 标准。整章涉及的“无线网络”“WLAN”等术语均指 802.11 标准中的网络。

空间上的分离是通过将频谱划分为不同信道实现的。一个信道只是 802.11 无线频谱的一部分。在美国，有 11 个信道可用（有些国家允许使用更多的信道）。这是很重要的，因为 WLAN 同时只能操作一个信道，就意味着我们只能同时嗅探一个信道，如图 13-1 所示。所以，如果你要处理信道 6 的 WLAN，就必须将系统配置成捕获信道 6 的流量。

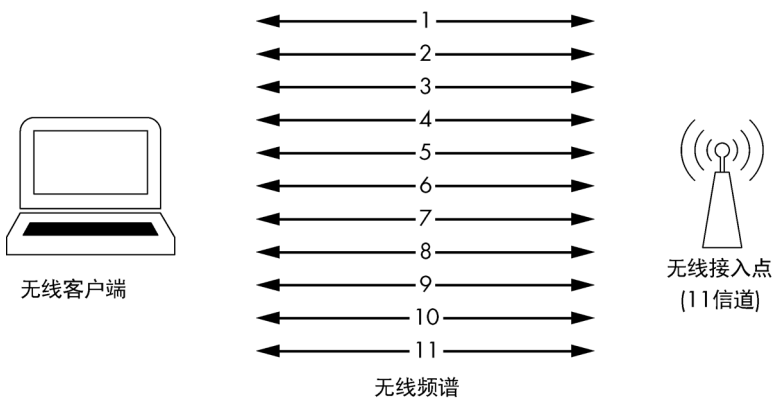


图 13-1 嗅探无线网络很麻烦，因为同一时间只能处理一个信道

异步社区Kensuke(18319066421) 专享 请尊重版权

注意 传统的无线嗅探只能同时处理一个信道，但有一个例外：某些无线扫描应用程序使用“跳频”技术，可以迅速改变监听信道以收集更多数据。其中一个很流行的工具是 Kismet 可以每秒跳跃 10 个信道，从而高效地嗅探多个信道。

13.1.2 无线信号干扰

当有其他因素干扰信号时，无线通信不能保证空气中传输的数据是完整的。无线网络有一定的抗干扰特性，但并不完全可靠。因此，当从无线网络捕获数据包时，你必须注意周边环境，确保没有大的干扰源，比如大型反射面、大块坚硬物体、微波炉、2.4GHz 无绳电话、厚墙面，以及高密度表面等。这些可能导致数据包丢失、重复数据包或数据包损坏。

同时你还要考虑信道间干扰。虽然同一时刻只能嗅探一个信道，但还是有些小小的忠告：无线频谱被分为多个不同的传输信道，但因为频谱空间有限，所以信道间有些许重叠，如图 13-2 所示。这意味着，如果信道 4 和信道 5 上都有流量，那么当你在其中一个信道上嗅探时，会捕获到另一个信道上的数据包。通常，同一地域上的多个网络被设置成使用 1、6 和 11 这 3 个不重叠信道，所以你可能不会遇到这个问题，但以防万一，你还是要了解这是怎么回事。

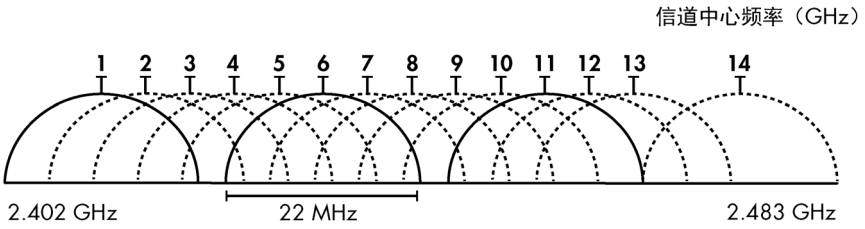


图 13-2 由于频谱空间有限，信道之间有重叠

13.1.3 检测和分析信号干扰

无线信号干扰的问题不是在 Wireshark 上观察数据包就能解决的。如果你致力于维护 WLAN，那么就应该定期检测信号干扰。这可以用频谱分析器来完成，它可以显示频谱上的数据或干扰。

商业的频谱分析器价格昂贵，价格甚至高达数千美元，但对于日常使用则有更好的方案。MetaGeek 开发了一个叫 Wi-Spy 的产品，这是一个 USB 硬件设备，

异步社区Kensuke(18319066421) 专享 请尊重版权

用于监测整个 802.11 频谱上的干扰。与 MetaGeek 的 Chanalyzer 软件搭配后，这个硬件可以输出图形化频谱，有助于解决无线网络的问题。Chanalyzer 的示例输出如图 13-3 所示。

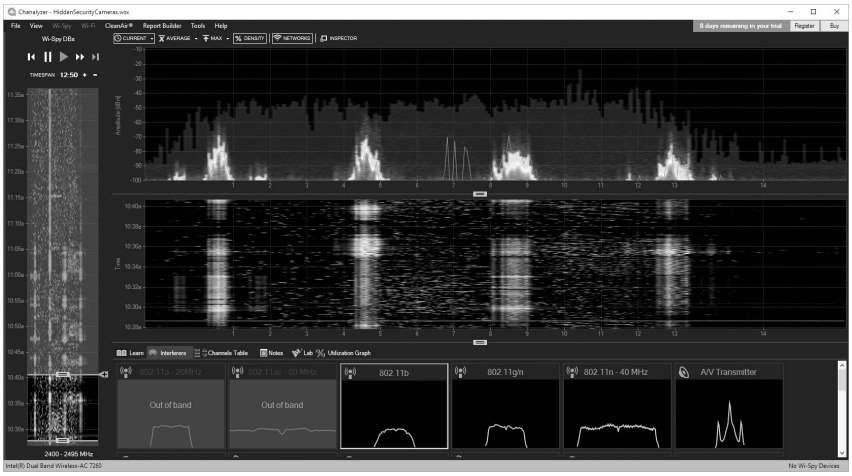


图 13-3　这个 Chanalyzer 显示同一地点有多个 WLAN 在工作

13.2　无线网卡模式

在开始嗅探无线数据包之前，我们需要了解无线网卡的不同工作模式。

无线网卡一共有 4 种工作模式。

被管理模式（Managed mode）：当你的无线客户端直接与无线接入点（Wireless Access Point，WAP）连接时，就可以使用这个模式。在这个模式中，无线网卡的驱动程序依赖 WAP 管理整个通信过程。

Ad-hoc 模式：当你的网络由互相直连的设备组成时，就可以使用这个模式。在这个模式中，无线通信双方共同承担 WAP 的职责。

主模式（Master mode）：一些高端无线网卡还支持主模式。这个模式允许无线网卡使用特制的驱动程序和软件工作，作为其他设备的 WAP。

监听模式（Monitor mode）：就我们的用途而言，这是一个重要的模式。当你希望无线客户端停止收发数据，专心监听空气中的数据包时，就可以使用监听模式。要使 Wireshark 捕获无线数据包，你的无线网卡和配套驱动程序必须支持监听模式（也叫 RFMON 模式）。

异步社区Kensuke(18319066421) 专享 请尊重版权

大部分用户只使用无线网卡的被管理模式或 Ad-hoc 模式。图 13-4 展示了各种模式如何工作。

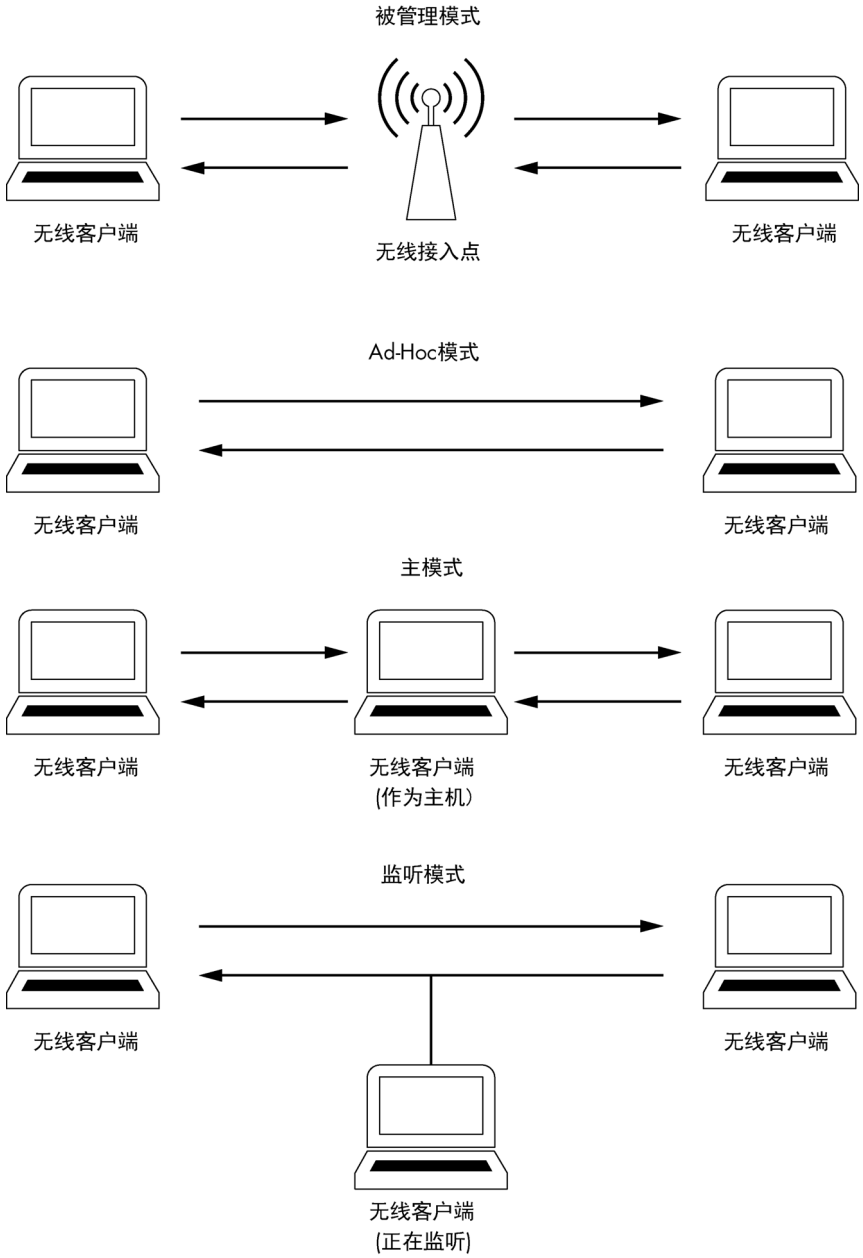


图 13-4　不同的无线网卡模式

异步社区Kensuke(18319066421) 专享 请尊重版权

注意　经常有人问我推荐哪款无线网卡做数据包分析。我强烈推荐自己使用的 ALFA 1000mW USB 无线适配器。这是市场上最好的产品之一，它能确保你捕获到每一个可能的数据包。大部分在线计算机硬件零售商都销售此款产品。

13.3　在 Windows 上嗅探无线网络

即使你有支持监听模式的无线网卡，大部分基于 Windows 的无线网卡驱动也不允许你切换到这个模式（WinPcap 也不支持这么做）。你需要一些额外的硬件来完成工作。

13.3.1　配置 AirPcap

AirPcap（现在是 Riverbed 旗下 CACE Technologies 公司的产品）被设计用来突破 Windows 强加给无线数据包分析的限制。AirPcap 像 U 盘一样小巧，如图 13-5 所示，用于捕获无线流量。AirPcap 使用第 3 章讨论的 WinPcap 驱动和一个特制的客户端配置工具。



图 13-5　AirPcap 的设计非常紧凑，适合与笔记本电脑一同携带

AirPcap 的配置程序很简单，只有一些配置选项。如图 13-6 所示，AirPcap 控制面板提供了以下几个选项。

Interface：你可以在这里选择要捕获的设备。一些高级的分析场景会要求你使用多个 AirPcap 设备，同步嗅探多个信道。

Blink Led：勾选这个复选框会使 AirPcap 设备上的 LED 指示灯闪烁。当存在多个 AirPcap 设备时，这可用来识别正在使用的适配器。

异步社区Kensuke(18319066421) 专享 请尊重版权

Channel：在这个下拉菜单里，你可以选择希望 AirPcap 监听的信道。

Include 802.11 FCS in Frames：默认情况下，一些系统会抛弃无线数据包的最后 4 个校验和比特。这个被称为帧校验序列（Frame Check Sequence，FCS）的校验和用来确保数据包在传输过程中没有被破坏。除非你有特别的理由，否则请勾选这个复选框（包含 FCS 校验和）。

Capture Type：这里有两个选项——802.11 Only 和 802.11 + Radio。802.11 Only 选项包含标准的 802.11 数据包头。802.11 + Radio 选项包含这个包头以及前端的 radiotap 头部，因而包含额外信息，比如数据率、频率、信号等级和噪声等级。选择 802.11 + Radio 以观察所有可用的数据信息。

FCS Filter：即便你没有选择 Include 802.11 FCS in Frames，这个选项也可以过滤 FCS 认为已经被损坏的数据包。使用 Valid Frames 选项可以只显示 FCS 认为成功接收的那些数据包。

WEP Configuration：这个区域（在 AirPcap Control Panel 的 Keys 选项卡可见）允许你输入所嗅探网络的 WEP 密码。为了能解密 WEP 加密的数据，你需要在这里填入正确的 WEP 密码。WEP 密码将在 13.9 节中讨论。

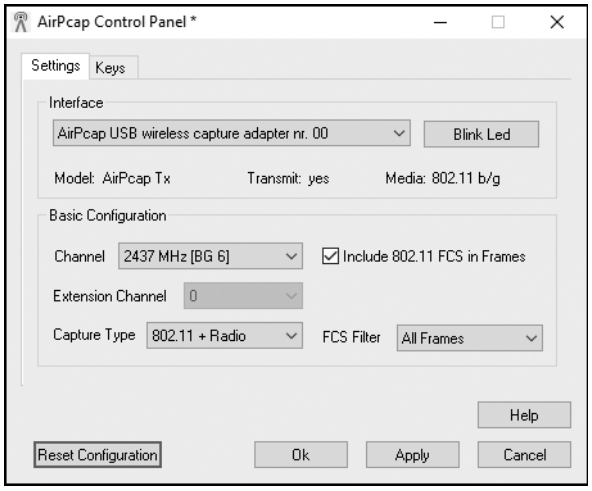


图 13-6　AirPcap 配置程序

13.3.2　使用 AirPcap 捕获流量

安装并配置好 AirPcap 后，你应该对捕获过程很熟悉了。只需要启动 Wireshark 并选择 **Capture->Options**。接着，在 **Interface** 下拉框中选择 AirPcap

异步社区Kensuke(18319066421) 专享 请尊重版权

设备❶，如图 13-7 所示。

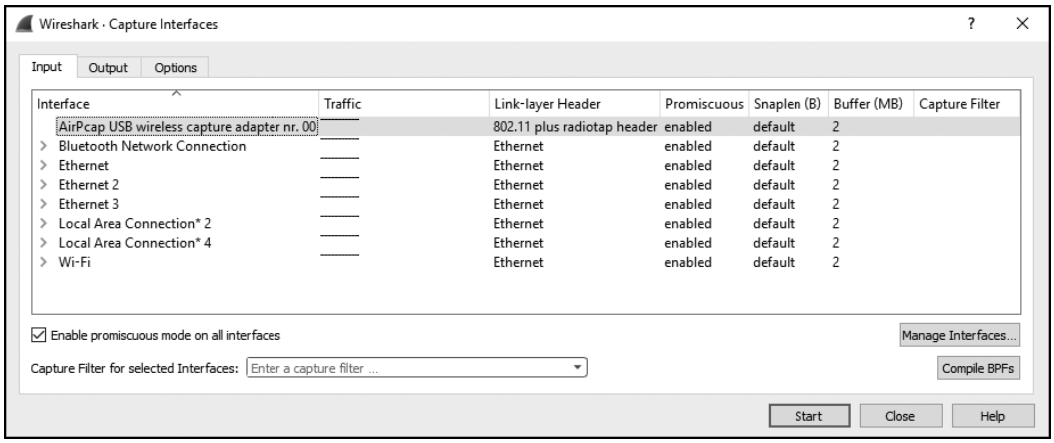


图 13-7　选择 AirPcap 设备作为你的捕获接口

除了 Wireless Settings 按钮外，屏幕上的一切都很熟悉。单击这个按钮会给出与 AirPcap 配置程序一样的选项，如图 13-8 所示。AirPcap 是完全嵌入 Wireshark 的，因此所有配置都可以在 Wireshark 中修改。

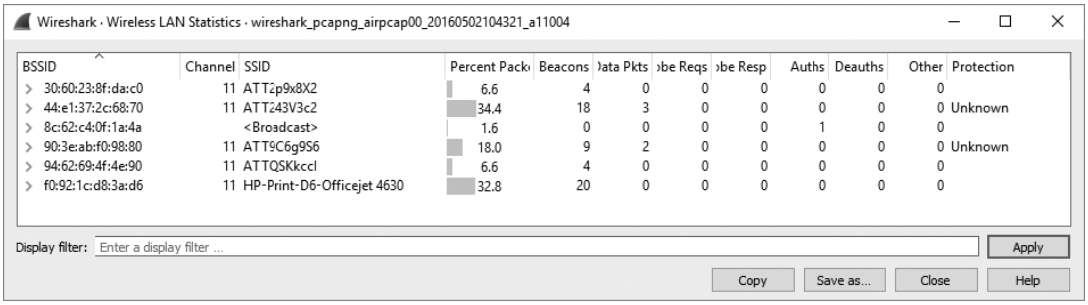


图 13-8　Advanced Wireless Settings 对话框允许你在 Wireshark 中配置 AirPcap

一切都配置好之后，就可以单击 Start 按钮开始捕获数据包了。

13.4　在 Linux 上嗅探无线网络

在 Linux 系统嗅探只需要简单地启用无线网卡的监听模式，然后启动 Wireshark 即可。然而，不同型号的无线网卡启用监听模式的流程各不相同，所以在这里我不能给出明确提示。实际上，有些无线网卡并不要求你启用监听模式。你最好 Google 一下你的网卡型号，确定是否需要启用它，以及如何启用。

异步社区Kensuke(18319066421) 专享 请尊重版权

在 Linux 系统中，通过内置的无线扩展程序启用监听模式是常用的办法之一。你可以用 iwconfig 命令打开无线扩展程序。如果你在控制台上键入 iwconfig，应该会看到这样的结果：

```
$ iwconfig
eth0  no wireless extensions
lo0   no wireless extensions
eth1  IEEE 802.11g     ESSID:"Tesla Wireless Network"
      Mode:Managed  Frequency:2.462 GHz  Access Point:00:02:2D:8B:70:2E
      Bit Rate:54 Mb/s  Tx-Power-20 dBm  Sensitivity=8/0
      Retry Limit:7  RTS thr: off  Fragment thr: off
      Power Management: off
      Link Quality=75/100  Signal level=-71 dBm  Noise level=-86 dBm
      Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
      Tx excessive retries:0  Invalid misc:0  Missed beacon:2
```

iwconfig 命令的输出显示 eth1 接口可以进行无线配置。这是显然的，因为它显示了与 802.11g 协议有关的数据，反观 eth0 和 lo0，它们只返回了“no wireless extensions”。

这个命令提供了许多无线配置信息，仔细看一下，有无线扩展服务设置 ID（Extended Service Set ID，ESSID）、频率等。我们注意到“eth1”下面一行显示，模式已经被设置为“被管理”，这也就是我们想改动的地方。

要将 eth1 改成监听模式，你必须以 root 用户身份登录。可以直接登录或用切换用户（su）命令，如下所示：

```
$ su
Password:<enter root password here>
```

在你成为 root 用户后，就可以键入命令来配置无线网卡选项了。输入以下命令可以将 eth1 配置成监听模式：

```
# iwconfig eth1 mode monitor
```

网卡进入监听模式后，再次运行 iwconfig 命令应该能反映出变化。输入以下命令，以确保 eth1 接口可以工作：

```
# iwconfig eth1 up
```

我们也将使用 iwconfig 命令改变监听信道，输入以下命令，改变 eth1 接口的信道为信道 3：

```
# iwconfig eth1 channel 3
```

注意 你可以在捕获数据包的过程中随意修改信道，所以随便改吧，没问题！也可以将 iwconfig 命令脚本化以简化过程。

异步社区Kensuke(18319066421) 专享 请尊重版权

完成这些配置后，请启动 Wireshark 开始你的数据包捕获之旅！

13.5 802.11 数据包结构

无线数据包与有线数据包的主要不同在于额外的 802.11 头部。这是一个第 2 层的头部，包含与数据包和传输介质有关的额外信息。802.11 分组有 3 种类型。

管理：这些分组用于在主机之间建立 2 层连接。管理分组还有一些重要的子类型，包括认证（authentication）、关联（association）和信号（beacon）分组。

控制：控制分组允许管理分组和数据分组的发送，并与拥塞管理有关。常见的子类型包括请求发送（request-to-send）和准予发送（clear-to-send）分组。

数据：这些分组含有真正的数据，也是唯一可以从无线网络转发到有线网络的数据包。

一个无线数据包的类型和子类型决定了它的结构，因此各种数据包结构可能不计其数。我们将考察其中一种结构，请看 80211beacon.pcap 文件里的单个数据包。这个文件包含一种叫 beacon 的管理数据包的例子，如图 13-9 所示。

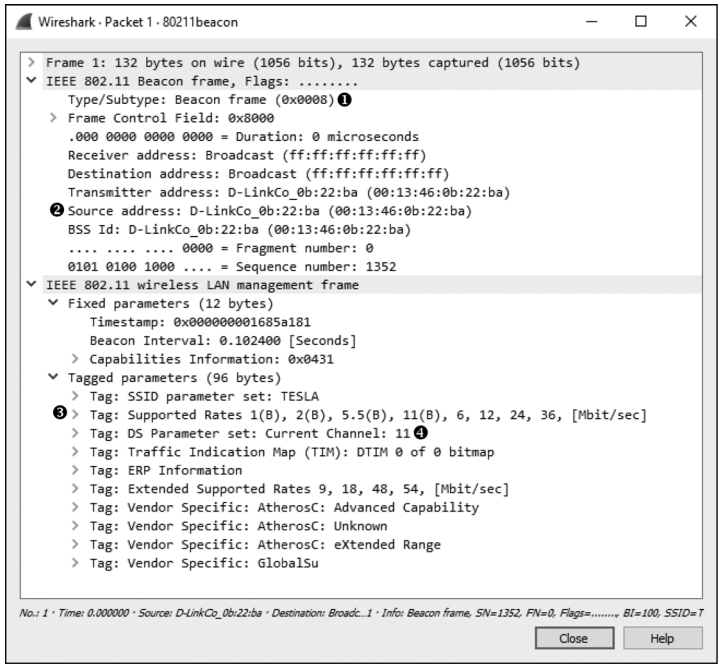


图 13-9　这是一个 802.11 beacon 数据包

异步社区Kensuke(18319066421) 专享 请尊重版权

beacon 是你能找到的最有信息量的无线数据包之一。它作为一个广播数据包由 WAP 发送，穿过无线信道通知所有无线客户端存在这个可用的 WAP，并定义了连接它必须设置的一些参数。在我们的示例文件中，你可以看到这个数据包在 802.11 头部的 Type/Subtype 域被定义为 beacon❶。

在 802.11 管理帧头部发现了其他信息，包括以下几点。

Timestamp：发送数据包的时间戳。

Beacon Interval：beacon 数据包重传间隔。

Capability Information：WAP 的硬件容量信息。

SSID Parameter Set：WAP 广播的 SSID（网络名称）。

Supported Rates：WAP 支持的数据传输率。

DS Parameter：WAP 广播使用的信道。

这个头部也包含了来源和目的地址以及厂商信息。

在这些知识的基础上，我们可以了解到示例文件中发送 beacon 的 WAP 的很多信息。显然这是一台 D-Link 设备❷，使用 802.11b 标准（B）❸，在信道 11 上工作❹。

虽然 802.11 管理数据包的具体内容和用途不一样，但总体结构跟这个例子相差不大。

13.6 在 Packet List 面板增加无线专用列

如你所见，Wireshark 通常在 Packet List 面板显示 6 个不同的列。分析无线数据包之前，让我们在 Packet List 面板增加 3 个新列。

- RSSI（for Received Signal Strength Indication）列，显示捕获数据包的射频信号强度。
- TX Rate（for Transmission Rate）列，显示捕获数据包的数据率。
- Frequency/Channel 列，显示捕获数据包的频率和信道。

当处理无线连接时，这些提示信息将会非常受用。例如，即使你的无线客户端软件告诉你信号强度很棒，捕获数据包并检查这些列时，也许会得到与该声明不符的数字。

异步社区Kensuke(18319066421) 专享 请尊重版权

按照以下步骤，在 Packet List 面板增加这些列。

（1）选择 **Edit->Preferences**。

（2）到 **Columns** 部分，并单击 **Add**。

（3）在 Title 域键入 **RSSI**，并在域类型下拉列表中选择 **IEEE 802.11 RSSI**。

（4）为 TX Rate 和 Frequency/Channel 列重复此过程，为它们取个恰当的 Title，并在 Field type 下拉列表选择 **IEEE 802.11 TX Rate** 和 **Channel/Frequency**。添加三列之后，Preferences 窗口应该像图 13-10 一样。

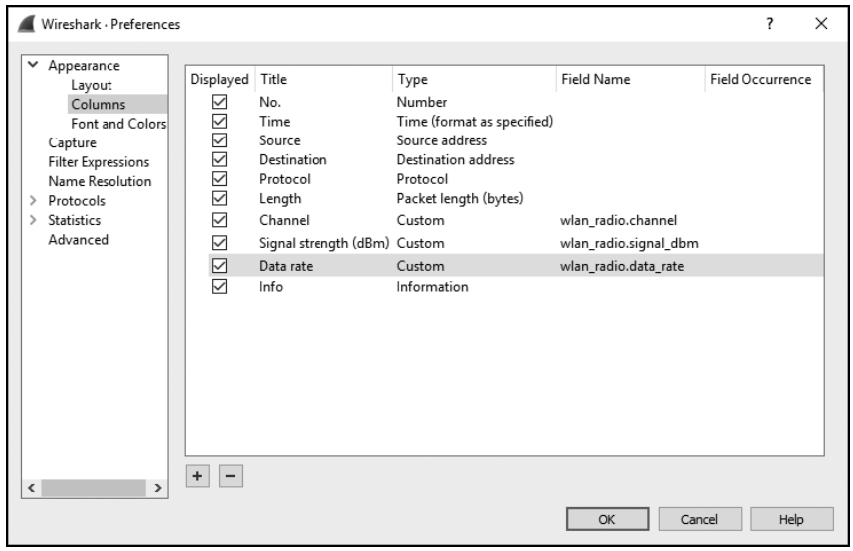


图 13-10　在 Packet List 面板增加与无线相关的列

（5）单击 **OK** 使改动生效。

（6）重启 Wireshark 以显示新列。

13.7　无线专用过滤器

我们在第 4 章讨论过了使用捕获和显示过滤器的好处。在有线网络中筛选流量要容易得多，因为每个设备有它自己的专用线缆。然而，在无线网络中，所有无线客户端产生的流量都同时存在于共享信道中，这意味着捕获任意一个信道，都将包含几十个客户端的流量。本节要讨论的焦点就是数据包过滤器，它可帮你找到特定流量。

异步社区Kensuke(18319066421) 专享 请尊重版权

13.7.1 筛选特定 BSS ID 的流量

网络上每一个 WAP 都有它自己的识别名，叫作“基础服务设备识别码”（Basic Service Set Identifier，BSS ID）。接入点发送的每一个管理分组和数据分组都包含这个名称。

一旦你明确了想要查看的 BSS ID 名称，那么你需要做的就只是找到那个 WAP 发送的数据包而已。Wireshark 在 Packet List 面板的 Info 列里显示了 WAP，因此找到这个信息易如反掌。

为了找到感兴趣的 WAP 所传输的数据包，你需要在 802.11 头部中找它的 BSS ID 域。过滤器就基于这个地址来编写。找到 BSS ID MAC 地址后，你可以使用这个过滤器：

```
wlan.bssid == 00:11:22:33:44:55
```

这样你就只能看见流经该特定 WAP 的流量了。

13.7.2 筛选特定的无线数据包类型

在本章前面，我们曾讨论过你可能在网络上看见的无线数据包类型。你通常需要基于这些类型和子类型来筛选数据包。对于特定类型，可以用过滤器 wlan.fc.type 来实现；对于特定类型或子类型的组合，可以用过滤器 wc.fc.type_subtype 来实现。例如，为了过滤一个 NULL 数据分组（在十六进制中是类型 2，子类型 4），你可以使用 wlan.fc.type_subtype eq 0x24 这个过滤器。表 13-1 提供了 802.11 分组类型和子类型的简要参考。

表 13-1　无线类型/子类型和相关过滤器语法

帧类型/子类型	过滤器语法
Management frame	wlan.fc.type == 0
Control frame	wlan.fc.type == 1
Data frame	wlan.fc.type == 2
Association request	wlan.fc.type_subtype == 0x00
Association response	wlan.fc.type_subtype == 0x01
Reassociation request	wlan.fc.type_subtype == 0x02
Reassociation response	wlan.fc.type_subtype == 0x03
Probe request	wlan.fc.type_subtype == 0x04
Probe response	wlan.fc.type_subtype == 0x05
Beacon	wlan.fc.type_subtype == 0x08

异步社区Kensuke(18319066421) 专享 请尊重版权

续表

帧类型/子类型	过滤器语法
Disassociate	wlan.fc.type_subtype == 0x0A
Authentication	wlan.fc.type_subtype == 0x0B
Deauthentication	wlan.fc.type_subtype == 0x0C
Action frame	wlan.fc.type_subtype == 0x0D
Block ACK requests	wlan.fc.type_subtype == 0x18
Block ACK	wlan.fc.type_subtype == 0x19
Power save poll	wlan.fc.type_subtype == 0x1A
Request to send	wlan.fc.type_subtype == 0x1B
Clear to send	wlan.fc.type_subtype == 0x1C
ACK	wlan.fc.type_subtype == 0x1D
Contention free period end	wlan.fc.type_subtype == 0x1E
NULL data	wlan.fc.type_subtype == 0x24
QoS data	wlan.fc.type_subtype == 0x28
Null QoS data	wlan.fc.type_subtype == 0x2C

13.7.3 筛选特定频率

如果你在查看来自多个信道的流量，那么基于信道的筛选就非常有用。例如，如果你本来只期待在信道 1 和 6 出现流量，那么可以输入一个过滤器显示信道 11 的流量。如果发现有流量，那么你就知道一定是哪里弄错了——或许是配置错误，或许有钓鱼无线设备。使用这个过滤器语法，可以筛选特定频率：

```
wlan_radio.channel == 11
```

这将显示信道 11 的所有流量。你可以将 2462 值替换成想筛选的信道对应的频率。表 13-2 列出了信道和频率的对应表格。

表 13-2 802.11 无线信道和频率

信道	频率/MHz
1	2412
2	2417
3	2422
4	2427
5	2432
6	2437
7	2442
8	2447

异步社区Kensuke(18319066421) 专享 请尊重版权

续表

信道	频率/MHz
9	2452
10	2457
11	2462

另外还有数百个实用的无线网络流量过滤器，你可以在 Wireshark wiki 上查看它们。

13.8 保存无线分析配置

在进行无线数据包分析时，预配置工作很麻烦——包括添加特定的栏目，编辑过滤器设置。为了避免每次都对栏目和过滤器进行重新配置，你可以将设置保存为配置文件，以便于在有线和无线分析配置之间切换。

在保存配置文件之前，先根据需要设置栏目和过滤器。之后，在屏幕右下角的当前配置列表上右击，并单击 **New**。将配置文件命名为 **Wireless**，最后单击 **OK**。

13.9 无线网络安全

部署和管理无线网络时最大的担忧就是传输数据的安全性。数据在空气中飞过，任何人都能得到它，因此数据加密是至关重要的。否则，任何人拿到 Wireshark 和 AirPcap 就都能看到数据了。

注意 当使用其他层次的加密技术时，比如 SSL 或 SSH，通信仍将在该层对数据进行加密，别人使用数据包嗅探器也读不到用户的通信内容。

最初推荐用在无线网络中加密传输数据的技术依据“有线等效加密”（Wired Equivalent Privacy，WEP）标准。WEP 在前几年很成功，直到后来发现了它在密钥管理方面的几个漏洞。为了加强安全，几个新标准又相继被设计出来，包括无线上网保护接入（Wi-Fi Protected Access，WPA）和 WPA2 标准。虽然 WPA 和更安全的版本 WPA2 仍然不可靠，但一般认为它们比 WEP 强多了。

异步社区Kensuke(18319066421) 专享 请尊重版权

在这一节，我们来看一些 WEP 和 WPA 流量，以及认证失败的例子。

13.9.1 成功的 WEP 认证

80211-WEPauth.pcap 文件包含了成功连接 WEP 无线网络的例子。这个网络使用 WEP 安全机制。你必须向 WAP 提供一个密码，以通过认证并解密它发来的数据。你可以把 WEP 密码当成无线网络密码。

如图 13-11 所示，这个捕获文件以数据包 4 所示的从 WAP（00:11:88:6b:68:30）发送到无线客户端（00:14:a5:30:b0:af）的质询开始❶。这个质询的目的是确认无线客户端是否有正确的 WEP 密码。展开 802.11 头部和 tagged parameters，你可以看到这个质询。

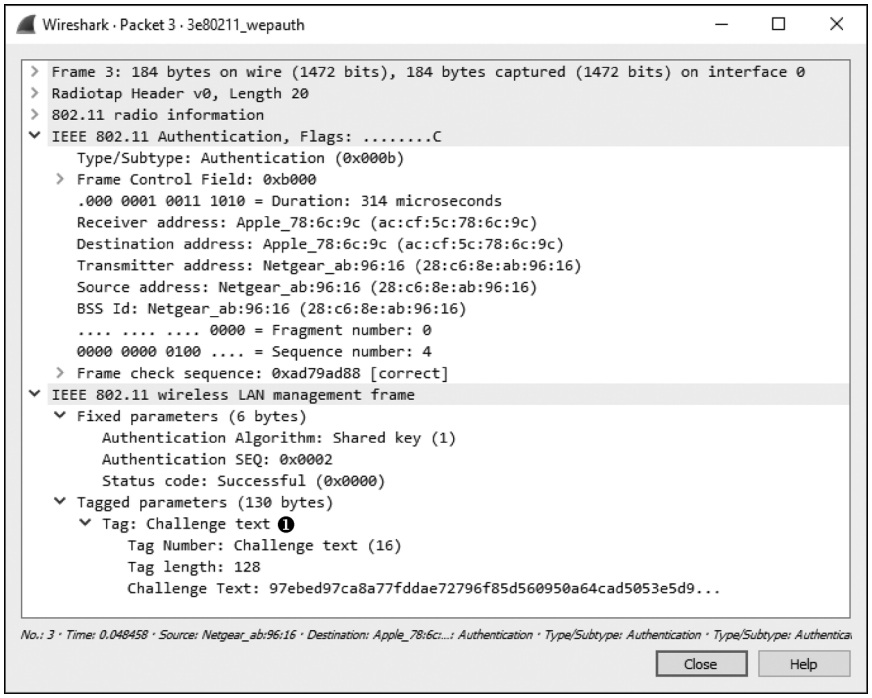


图 13-11　WAP 给无线客户端发送质询文本

在数据包 5 中，这个质询被确认。然后无线客户端将用 WEP 密码解密的质询文本返回给 WAP❶，如图 13-12 所示。

在数据包 7 中，这个数据包被再次确认，并且 WAP 在数据包 8 中响应了无线客户端，如图 13-13 所示。响应里包含了一个说明认证成功的通知❶。

异步社区Kensuke(18319066421) 专享 请尊重版权

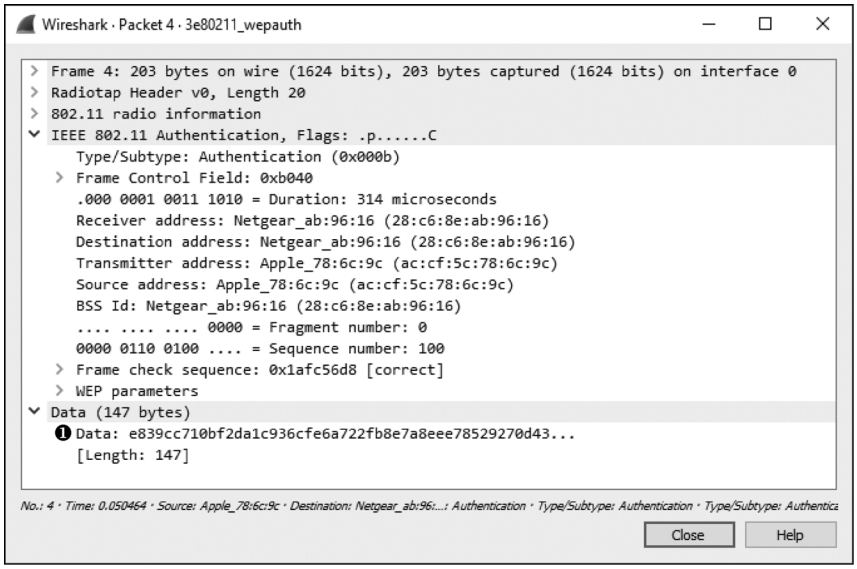


图 13-12　无线客户端向 WAP 发送已解密的质询文本

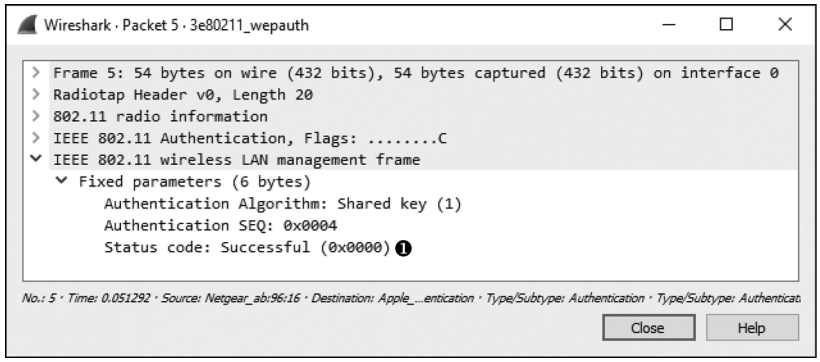


图 13-13　WAP 通知客户端认证成功了

成功认证后，客户端可以发送关联（association）请求、接收确认并完成连接过程，如图 13-14 所示。

No.	Time	Source	Destination	Protocol	Length	Channel	Signal strength (dBm)	Data rate	Info
6	0.052565	Apple_78:6c:9c	Netgear_ab:96:16	802.11	110	1	-40	1	Association Request, SN=101, FN=0, Flags=........C, SSID=DENVEROFFICE
7	0.053902	Netgear_ab:96:16	Apple_78:6c:9c	802.11	119	1	-17	1	Association Response, SN=6, FN=0, Flags=........C

图 13-14　认证过程后紧跟一个简单的双数据包关联请求和响应

13.9.2　失败的 WEP 认证

在下一个例子中，一位用户输入他的 WEP 密码连接到 WAP，几秒后，无

异步社区Kensuke(18319066421) 专享 请尊重版权

线客户端程序报告无法连接到无线网络，但没有给出原因。捕获的文件是80211-WEPauthfail.pcap。

与成功连接时一样，通信从 WAP 在数据包 3 发送质询文本到无线客户端开始。这个消息被成功确认了。接着，在数据包 4 中，无线客户端使用用户提供的 WEP 密码发送了响应。

到这里，我们会想，应该有一个通知告诉我们认证成功了。但是我们在数据包 5 却看到了不一样的情况，如图 13-15 所示❶。

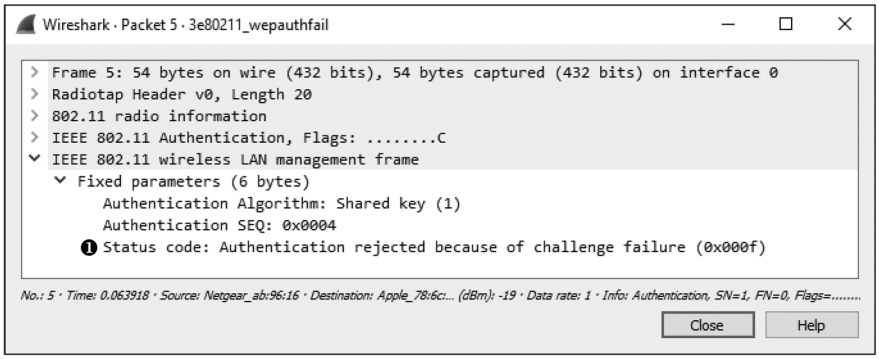


图 13-15　这个消息告诉我们认证不成功

这个消息告诉我们无线客户端对质询文本的响应不正确。这表明客户端用以解密质询文本的 WEP 密码肯定输错了。结果，连接过程就失败了。必须用正确的 WEP 密码重试才行。

13.9.3　成功的 WPA 认证

WPA 使用了与 WEP 完全不同的认证机制，但它仍然依赖于用户在无线客户端输入的密码来连接到网络。80211-WPAauth.pcap 文件中有一个成功的 WPA 认证的例子。

该文件的第 1 个数据包是 WAP 发送的 beacon 广播。我们展开这个数据包的 802.11 头部，沿着 tagged parameters 往下看，展开 Vendor Specific 标题，如图 13-16 所示，能看到无线接入点的 WPA❶属性部分。这让我们了解到 WAP 支持的 WPA 版本与实现（如果有的话）。

无线客户端（00:0f:b5:88:ac:82）收到这个 beacon 广播后，就向无线接入点（00:14:6c:7e:40:80）发送一个探测请求，并得到了响应。无线客户端和无线接入

异步社区Kensuke(18319066421) 专享 请尊重版权

点在数据包 4 到 7 之间的交互，是认证与关联的请求及响应。

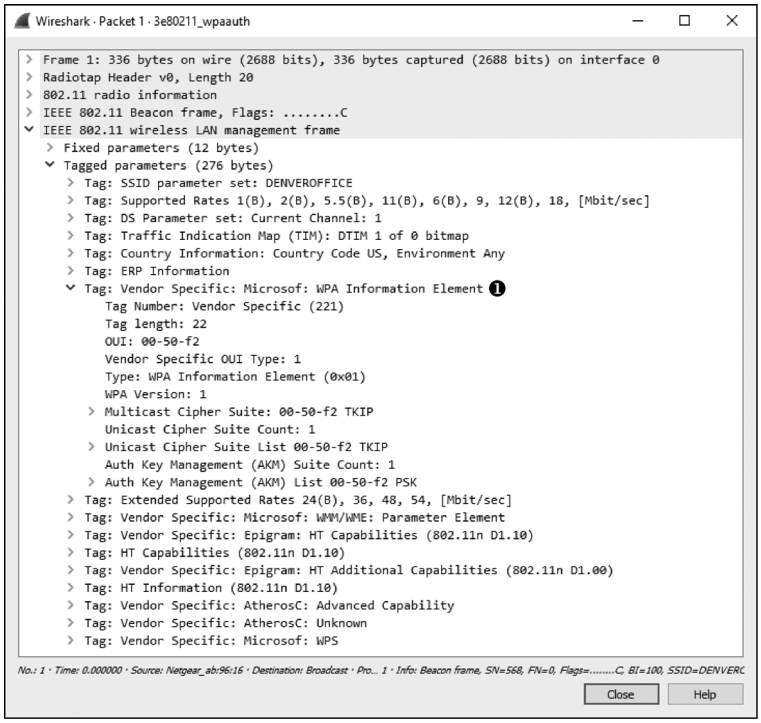


图 13-16　这个 beacon 让我们知道无线接入点支持 WPA 认证

现在把目光转移到数据包 8 上。这是 WPA 开始握手的地方，一直持续到数据包 11。这个握手过程就是 WPA 质询响应的过程，如图 13-17 所示。

No.	Time	Source	Destination	Protocol	Length	Channel	Signal strength (dBm)	Data rate	Info
8	0.…	Netgear_ab:96:16	Apple_78:6c:9c	EAPOL	157	1	-18	24	Key (Message 1 of 4)
9	0.…	Apple_78:6c:9c	Netgear_ab:96:16	EAPOL	183	1	-42	1	Key (Message 2 of 4)
10	0.…	Netgear_ab:96:16	Apple_78:6c:9c	EAPOL	181	1	-18	36	Key (Message 3 of 4)
11	0.…	Apple_78:6c:9c	Netgear_ab:96:16	EAPOL	157	1	-42	1	Key (Message 4 of 4)

图 13-17　这些数据包是 WPA 握手的一部分

这里有两个质询与响应。每个数据包都可在基于 802.1x Authentication 头部下的 Replay Counter 域找到匹配对象，如图 13-18 所示。注意到前两个握手数据包的 Replay Counter 值是 1❶，而后两个握手数据包的值是 2❷。

WPA 握手完成并认证成功后，数据就开始在无线客户端和 WAP 之间传输了。

异步社区Kensuke(18319066421) 专享 请尊重版权

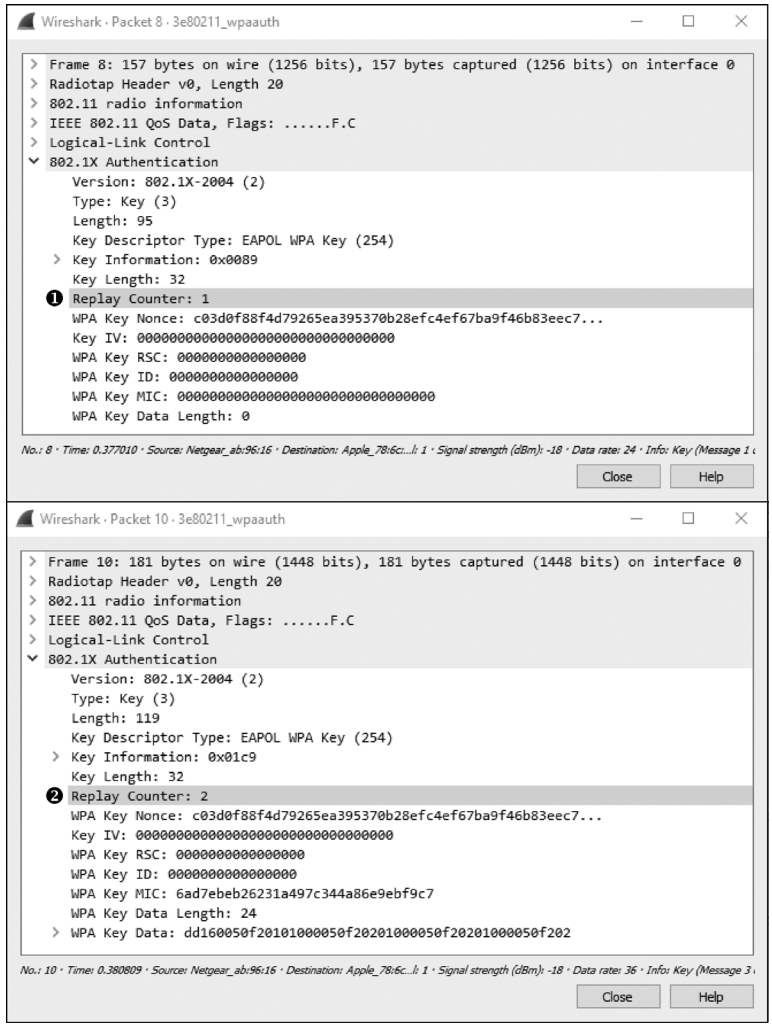


图 13-18　Replay Counter 域帮助我们匹配质询和响应

13.9.4　失败的 WPA 认证

与 WEP 一样，用户输入 WPA 密码后，无线客户端程序报告无法连接到无线网络，但没有指出问题在哪里，我们来看一看发生了什么。捕获的结果保存在 80211-WPAauthfail.pcap 文件中。

像刚才成功的 WPA 认证那样，捕获文件以同样的方式开始。这包括探测、

异步社区Kensuke(18319066421) 专享 请尊重版权

认证和关联请求。WPA 握手从数据包 8 开始，但在这个例子中，我们看到了 8 个握手数据包，而不是之前在成功认证环节中看到的 4 个。

数据包 8 和 9 表示 WPA 握手的前两个数据包。然而在这个例子中，客户端发送回 WAP 的质询文本有误。结果，这个序列在数据包 10 和 11、12 和 13、14 和 15 中多次重复，如图 13-19 所示。使用 Replay Counter 可以配对每个请求和响应。

No.	Time	Source	Destination	Protocol	Length	Channel	Signal strength (dBm)	Data rate	Info
8	0.073773	Netgear_ab:96:16	Apple_78:6c:9c	EAPOL	157	1	-18	24	Key (Message 1 of 4)
9	0.076510	Apple_78:6c:9c	Netgear_ab:96:16	EAPOL	183	1	-30	1	Key (Message 2 of 4)
10	1.074290	Netgear_ab:96:16	Apple_78:6c:9c	EAPOL	157	1	-19	24	Key (Message 1 of 4)
11	1.076573	Apple_78:6c:9c	Netgear_ab:96:16	EAPOL	183	1	-32	1	Key (Message 2 of 4)
12	2.075292	Netgear_ab:96:16	Apple_78:6c:9c	EAPOL	157	1	-18	36	Key (Message 1 of 4)
13	2.077610	Apple_78:6c:9c	Netgear_ab:96:16	EAPOL	183	1	-29	1	Key (Message 2 of 4)
14	3.077211	Netgear_ab:96:16	Apple_78:6c:9c	EAPOL	157	1	-18	48	Key (Message 1 of 4)
15	3.079537	Apple_78:6c:9c	Netgear_ab:96:16	EAPOL	183	1	-32	1	Key (Message 2 of 4)

图 13-19　这里的额外 EAPOL 数据包表明 WPA 认证失败了

握手过程重试 4 次后，通信中止了。如图 13-20 所示，数据包 16 表明无线客户端没有通过认证❶。

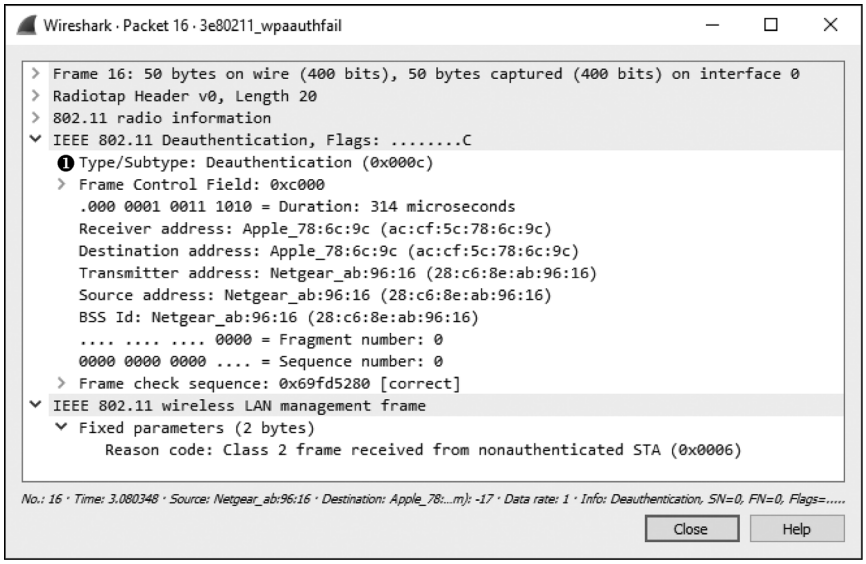


图 13-20　WPA 握手失败后，客户端认证失败

异步社区Kensuke(18319066421) 专享 请尊重版权

13.10 小结

虽然无线网络仍然被普遍认为是不安全的，但它在各个组织环境的部署却丝毫没有减缓。随着人们将焦点转移到无线通信，掌握类似有线网络那样的捕获、分析方法，并用于无线网络数据包变得尤为重要。当然，本章讲授的概念和技巧并不全面，但它们能帮助你使用数据包分析技术解决无线网络问题，让你赢在起跑线上。

异步社区Kensuke(18319066421) 专享 请尊重版权

附录 A

延伸阅读



数据包分析工具不只有 Wireshark，还有一大堆趁手的工具，可以在解决网络缓慢、网络安全等常规问题及分析无线网络问题时大显威力。本章列出了一些有用的数据包分析工具，以及其他数据包分析的学习资源。

A.1 数据包分析工具

除了 Wireshark 之外，还有一些实用的数据包分析工具。在这里，我会介绍一些我认为最有用的。

异步社区Kensuke(18319066421) 专享 请尊重版权

1. Tcpdump 和 Windump

虽然 Wireshark 很流行，但它可能没有 Tcpdump 用得广泛。考虑到一些人群对数据包捕获和分析的实际需求，Tcpdump 是完全基于文本的。

虽然 Tcpdump 缺少图形特性，但它处理海量数据时非常靠谱。因为你可以用管道将它的输出重定向输入给其他命令，比如 Linux 的 sed 和 awk。随着对数据包分析的深入钻研，你会发现 Wireshark 和 Tcpdump 都很有用。

Windump 只是 Tcpdump 在 Windows 平台的发行版而已。

2. Cain & Abel

第 2 章已经讨论过，Cain & Abel 是 Windows 平台上最好的 ARP 缓存中毒攻击工具之一。Cain & Abel 实际上是一个非常健壮的工具套件，你一定能发现其他用途。

3. Scapy

Scapy 是一个非常强大的 Python 库，允许使用基于命令行脚本的方法创建、修改数据包。简单地说，Scapy 是一款强大、灵活的数据包操纵程序。

4. Netdude

如果你不需要像 Scapy 那样高级的工具，那么 Netdude 是 Linux 下的一个较好的替代品。虽然 Netdude 功能有限，但它提供了图形用户界面，因而出于研究目的，需要创建、修改数据包时，它显得极其方便。图 A-1 演示了使用 Netdude 的一个例子。

5. Colasoft Packet Builder

如果你是 Windows 用户，并且想要与 Netdude 类似的 GUI，那么你可以考虑使用 Colasoft Packet Builder，一款超棒的免费工具。Colasoft 还提供了一个简便的用于数据包创建和修改的 GUI。

6. CloudShark

CloudShark（由 QA Café 开发）是我很喜爱的一个工具，可以用它在线分享数据包捕获记录。如图 A-2 所示，CloudShark 网站可以在浏览器里以 Wireshark 的方式显示网络捕获文件。你可以上传捕获文件，并将链接发送给同事，以便共同分析。

异步社区Kensuke(18319066421) 专享 请尊重版权

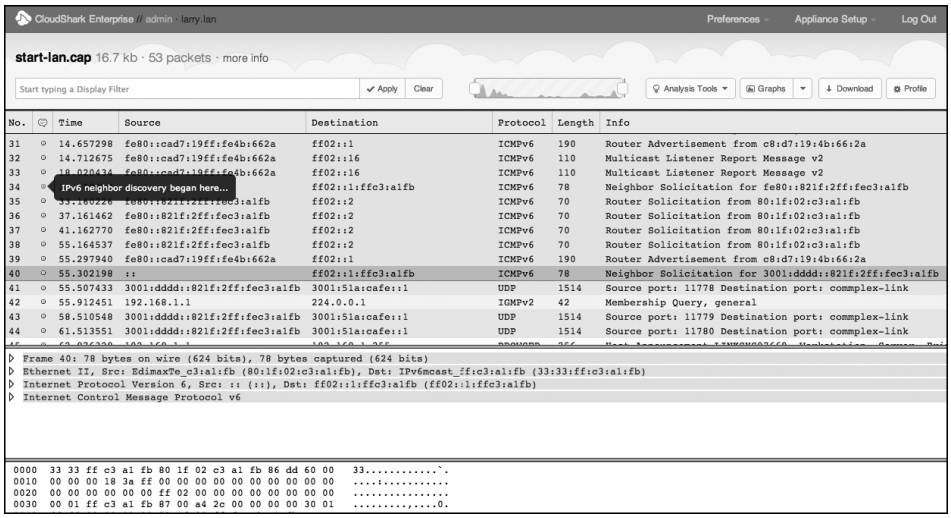


图 A-1　在 Netdude 上修改数据包

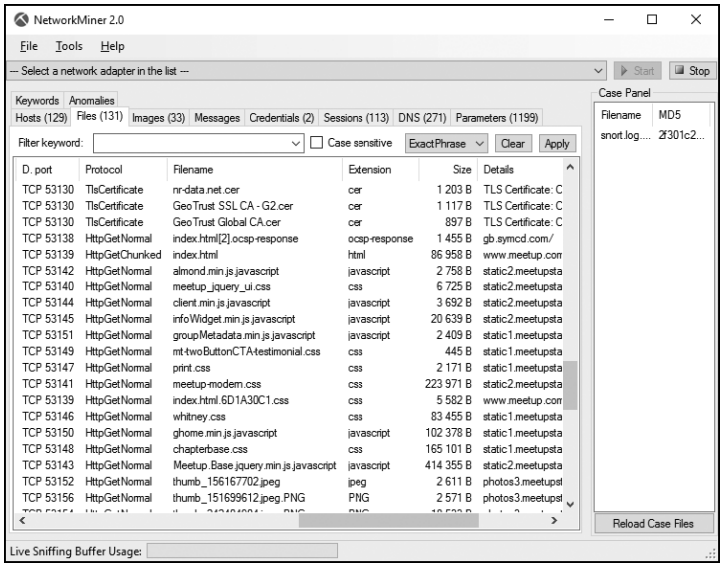


图 A-2　用 CloudShark 查看一个捕获文件示例

关于 CloudShark，我最赞赏的是它不需要注册，并能通过 URL 直接链接获取。这意味着，当我在博客上发布一个 PCAP 文件的链接时，其他人只需要单击就能查看数据包，而不需要在下载文件后，再用 Wireshark 打开。

异步社区Kensuke(18319066421) 专享 请尊重版权

7. pcapr

pcapr 是 Mu Dynamics 创建的一个非常健壮的用于分享 PCAP 文件的 Web 2.0 平台。在撰写本文时，pcapr 包含了将近 3000 个 PCAP 文件，涉及 400 多种不同协议的例子。图 A-3 显示了 pcapr 上的 DHCP 流量捕获的例子。

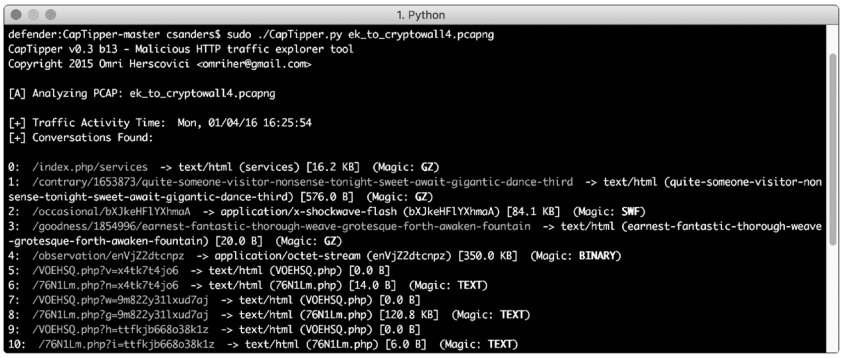


图 A-3　在 pcapr 上查看 DHCP 流量捕获

每次要查找某种确定类型的通信样例时，我都是首先在 pcapr 上搜索。如果你在自己的试验中创建了大量不同的捕获文件，不要犹豫，请将它们上传到 pcapr 社区分享。

8. NetworkMiner

NetworkMiner 是一款主要用于网络取证的工具，但我发现它在其他一些情形下也非常实用。虽然它也可以用来捕获数据包，但它的强项在于如何解析数据包。NetworkMiner 会检测 PCAP 文件中网络各端的操作系统类型，并将文件解析成主机间的会话。它甚至允许你直接从捕获记录中提取传输的文件。

9. Tcpreplay

每当有一堆数据包需要在线路上重传以观察设备如何响应它们时，我就用 Tcpreplay 来执行这个任务。Tcpreplay 专门设计用来重传 PCAP 文件里的数据包。

10. ngrep

如果你熟悉 Linux，毫无疑问，你肯定用过 grep 搜索数据。ngrep 与它非常相似，允许你在 PCAP 数据上执行特定搜索。当捕获和显示过滤器都无法实现我的目标或者实现太复杂时，我就使用 ngrep。

异步社区Kensuke(18319066421) 专享 请尊重版权

11. Libpcap

如果你计划开发一款应用程序，来进行一些确实高级的数据包解析，或是创建处理数据包，那么你要对 Libpcap 非常熟悉。简言之，Libpcap 是一个用于网络流量捕获的可移植的 C/C++库。Wireshark、Tcpdump，以及其他大部分数据包分析工具都在一定层次上依赖于 Libpcap。

12. Hping

Hping 是你武器库中应有的“瑞士军刀”之一。Hping 是一个命令行的数据包操纵和传输工具。它支持各种各样的协议，反应非常快且直观。

13. Domain Dossier

如果你需要查询域名或 IP 地址的注册信息，那么 Domain Dossier 正合你意。它快速、简单、有效。

14. Perl 和 Python

Perl 和 Python 虽然不是工具，但却是值得留意的脚本语言。当你熟练于数据包分析时，你会遇到没有自动化工具满足要求的情况。在那些情况下，首选 Perl 和 Python 语言编写工具，它们可以带你在数据包上做些有趣的事情。对于大部分应用程序，我通常使用 Python，但这只是个人选择。

A.2 数据包分析资源

从 Wireshark 的主页到教程、博客，有很多可用的数据包分析资源。我将在此列出我最喜欢的一些。

1. Wireshark 主页

与 Wireshark 有关的首要资源就是它的主页。主页包括软件文档、一个非常有用的包含了捕获文件样例的 wiki，以及 Wireshark 邮件列表的注册信息。

2. SANS 安全入侵检测深入课程

作为一名 SANS 导师，我可能会有点偏袒，但我真不认为这个星球上有比《SANS SEC 503：深度入侵检测》更好的数据包分析课程。这个课程集中于数据包分析的安全方面。即便你不集中于安全，该课程之前提供的对数据包分析和对 Tcpdump 的介绍也是我所见最好的。

异步社区Kensuke(18319066421) 专享 请尊重版权

该课程由我的两位数据包分析英雄 Mike Poor 和 Judy Novak 讲授。它每年提供好几次直播。若你的旅行经费有限，没关系，该课程也通过基于 Web 的按需格式在线讲授。

3. Chris Sanders 的博客

我没有太多时间写博客，但偶尔也会在我的博客上写一些有关数据包分析的文章。如果没有别的，我的博客就作为链接到我写的其他文章和书籍的门户，另外它也提供了我的联系方式。

4. Brad Duncan 的恶意软件流量分析网站

我最喜欢的安全相关数据包捕获资源是 Brad Duncan 的恶意软件流量分析（MTA）网站。Brad 每周多次发布包含感染链的数据包捕获资源，这些捕获资源包含相关的恶意软件二进制文件以及正在发生的事件的描述。如果你想获得解析恶意软件感染的经验并了解当前的恶意软件技术，请先下载其中一些捕获资源并尝试理解它们，你可以访问该网站，以便在发布更新时收到提醒。

5. IANA

互联网编号分配机构（Internet Assigned Numbers Authority，IANA）负责监督为北美分配 IP 地址和协议号码。它的网站提供了一些有价值的参考工具，比如查找端口号、查看有关顶级域名的信息，以及浏览合作以网站查阅 RFC 文档。

6.《TCP/IP 详解》（Addison-Wesley）

对生活在数据包层次的人而言，Richard Stevens 博士撰写的系列书籍是书架上的主要书目，已被多数人奉为 TCP/IP 圣经。这是我最喜欢的 TCP/IP 书籍，也是我写作本书时经常参考的文献。

7.《TCP/IP 指南》（No Starch 出版社）

在 TCP/IP 领域里，我最喜欢的另一本书是 Charles M. Kozierok 写的。这本巨著厚达 1000 多页，内容非常详细，并且为视觉型学习者准备了大量很优秀的图表。

异步社区Kensuke(18319066421) 专享 请尊重版权

附录 B

分析数据包结构



在本附录中，我们将讨论数据包的表现形式。我们将查看被解析为十六进制形式的数据包，另外，我们将介绍如何使用数据包结构图查看和表示数据包。

由于有大量的软件为你解析数据包，因此在数据包嗅探和分析时，你不需要理解本附录中的内容。但是，如果你投入一些时间学习数据包原始数据及其结构，会更好地理解 Wireshark 之类的工具展示给你的内容。在分析数据时，数据抽象程度越低越好。

B.1 数据包表现形式

数据包能够以很多表现形式被解析。数据包原始数据可以表现为二进制数据——二进制的 0、1 序列，如：

异步社区Kensuke(18319066421) 专享 请尊重版权

```
01100000010100110101110000001010110000010000000000010000000000000010001100000010
11010101101110000000000000000000000000000000000000000000010000001000000000001011
01101000000000000010000001100000000000000000000000010000000100000000010000000010
```

二进制数码是数据信息在最底层的表现形式，1 表示高电平信号，0 表示低电平信号。每一个数字是一位，八位是一字节。然而，人们很难阅读和理解二进制数据，所以我们通常将二进制数据转换为十六进制——由字母和数字组成的十六进制数据，如：

```
4500 0034 40f2 4000 8006 535c ac10 1080
4a7d 5f68 0646 0050 7c23 5ab7 0000 0000
8002 2000 0b30 0000 0204 05b4 0103 0302
0101 0402
```

十六进制（通常简写为 hex）是使用数字 0～9 和字母 A～F 表示数值的计数系统。十六进制是常用的数据包表示形式，因为其形式简洁，并且很容易被转换为更基础的二进制。在十六进制中，两个字符表示一个字节，即八位。字节中的每个字符是一个半字节（4 位），左侧的值是高位（半）字节，右侧的值是低位（半）字节。在示例数据包中，第一个字节为 45，其中，高位字节是 4，低位字节是 5。

数据包中，字节的地址（或位置）使用偏移量表示法表达，从 0 开始。因此，数据包中的第一个字节（45）位于 0x00，第二个字节（00）位于 0x01，第三个字节（00）位于 0x02，依此类推。0x 说明使用的是十六进制表示法。在表示一个长度大于一字节的地址时，地址占用的字节数在一个冒号之后使用数值表示。例如，在表示示例数据包的前 4 字节的地址时，使用 0x00:4 表示。在我们后续使用数据包结构图剖析“驾驭一个神秘的数据包”中的未知协议时，此处的说明非常重要。

注意

据我观察，人们在分析数据包时普遍的错误是忘了从 0 开始计算地址。的确很难习惯从 0 开始计算，因为大部分人学习的都是从 1 开始计数。我已经分析数据包很多年了，但是仍然会犯这样的错。我能给出的最好的建议是，不要害怕掰手指数数。你也许会觉得这样看起来很愚蠢，但是这其实没什么丢人的，尤其是在能帮你得出正确答案的情况下。

在更高的层级，一个与 Wireshark 类似的工具使用协议分析器将数据包使用完全解析的形式展示出来，我们后续将介绍此工具。本例中的数据包被 Wireshark 解析后，如图 B-1 所示。

异步社区Kensuke(18319066421) 专享 请尊重版权

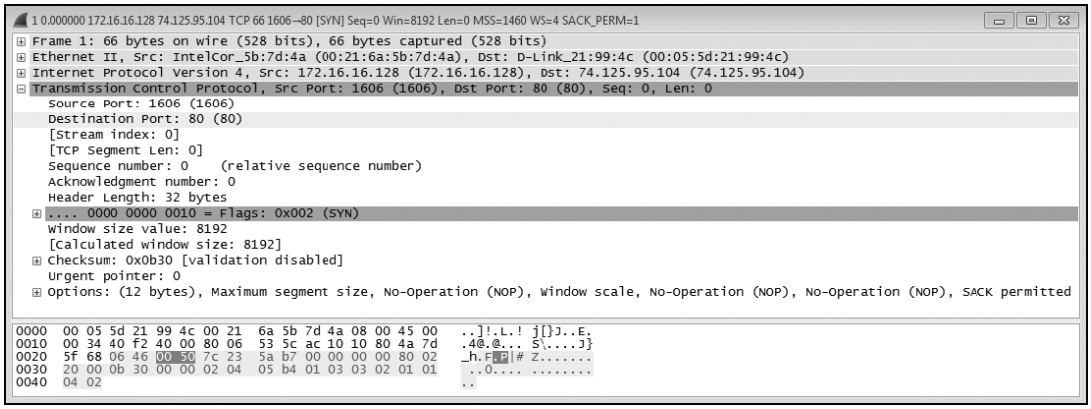


图 B-1　Wireshark 解析后的数据包

Wireshark 显示数据包信息，并为各字段添加标签。原始数据包并不包含标签，但是其中的数据按照协议标准规定的明确格式排列。完全解析数据包意味着将数据按照协议标准分析为带有标签的、可阅读的文本。

Wireshark 及类似的工具能够完全解析数据包，因为它们的内置协议分析器对协议各字段的地址、长度和值进行了定义。例如，图 B-1 中的数据包按照传输控制协议（TCP）标准分成多个部分，包括带有标签的字段和值。其中一个标签是源端口（Source Port），值为十进制的 1606。这使你在分析数据包时能够轻易找到指定信息。当你能够使用此类工具时，它们会成为你完成分析工作的一个高效的方式。

Wireshark 有成百上千个协议分析器，但是你仍有可能遇到 Wireshark 无法解析的协议；厂商定制的未广泛使用的协议和定制的恶意软件协议经常会是这种情况。当这样的事情发生时，数据包中只有部分内容能够被解析。这也是 Wireshark 默认在界面下方提供原始的十六进制包数据的原因（如图 B-1 所示）。

更普遍的情况是，如 Tcpdump 的命令行程序不提供太多的协议分析器，而是显示大量原始十六进制数据。对于一些更复杂的应用层协议而言，这种情况尤为常见，因为这类协议很难解析。因此，当我们使用 Tcpdump 时，看到被部分解析的数据包是常态。一个使用 Tcpdump 分析数据包的例子如图 B-2 所示。

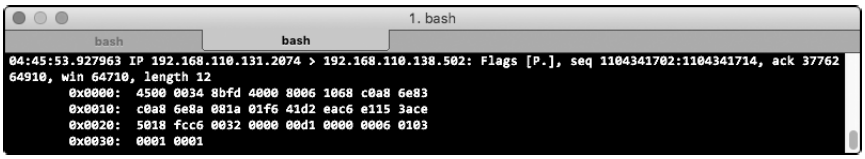


图 B-2　Tcpdump 中部分解析的数据包

异步社区Kensuke(18319066421) 专享 请尊重版权

当面对部分解析的数据包时，你需使用更底层的数据包结构知识。Wireshark、Tcpdump 及大部分其他工具提供了十六进制的原始包数据，帮助我们进行更底层的分析工作。

B.2 使用数据包结构图

我们在第 1 章中学习到，数据包是按照协议规定的方式排列的数据。因为通用协议按照指定的规则排列包数据，所以软硬件能够解析这些数据；数据包必须遵守明确的格式规则。使用数据包结构图，我们能够识别格式并解析数据包。一个数据包结构图是数据包的图形化表现方式，使分析人员能够将任意给定协议的数据包从原始十六进制字节映射至具体字段。结构图从协议的 RFC 文档中提取，显示了协议中各字段的长度和排列顺序。

让我们查看第 6 章中的 IPv4 的数据包结构图（见图 B-3）。

<table>
<tr><td colspan="10">互联网协议 4 (IPv4)</td></tr>
<tr><td>偏移位</td><td>八位组</td><td colspan="2">0</td><td colspan="2">1</td><td colspan="2">2</td><td colspan="2">3</td></tr>
<tr><td>八位组</td><td>位</td><td>0–3</td><td>4–7</td><td colspan="2">8–15</td><td>16–18</td><td>19–23</td><td colspan="2">24–31</td></tr>
<tr><td>0</td><td>0</td><td>版本号</td><td>首部长度</td><td colspan="2">服务类型</td><td colspan="3">总长度</td></tr>
<tr><td>4</td><td>32</td><td colspan="4">标识符</td><td>标识</td><td colspan="3">分片偏移</td></tr>
<tr><td>8</td><td>64</td><td colspan="3">存活时间</td><td colspan="2">协议</td><td colspan="3">首部校验和</td></tr>
<tr><td>12</td><td>96</td><td colspan="9">源IP地址</td></tr>
<tr><td>16</td><td>128</td><td colspan="9">目的IP地址</td></tr>
<tr><td>20</td><td>160</td><td colspan="9">选项</td></tr>
<tr><td>24+</td><td>192+</td><td colspan="9">数据</td></tr>
</table>

图 B-3　IPv4 的数据包结构图

在这个结构图中横轴表示 0～31 的二进制位。换算成字节，为 0～3。纵轴也按照位和字节进行标记，每行为一个 32 位（或 4 字节）片段。我们使用数轴来计算字段地址的偏移量，根据纵轴确定字段位于哪个 4 字节片段，然后根据横轴确定给字段以字节为单位的偏移量。第一行由前四个字节组成，0～3，在横轴上进行标记。第二行由之后的四字节组成， 4～7，同样在横轴上进行计数。我们从字节 4 开始计数，对应的横轴刻度为 0；下一个字节是字节 5，对应的横轴刻度为 1；以此类推。

例如，我们查看 IPv4 的结构图，0x01 字节是服务类型字段。计算方式：在

异步社区Kensuke(18319066421) 专享 请尊重版权

纵轴上看，服务类型字段位于第一行，对应纵轴刻度为 0，所以从 0 开始计数；在横轴上看，该字段位于刻度 1；所以该字段位于，从 0 开始且偏移量为 1 字节的位置，即位于 0x01 字节。

再查看另一个例子，0x08 字节是存活时间字段。计算方式：在纵轴上看，存活时间字段位于第三行，对应纵轴刻度为 8，所以从 8 开始计数；在横轴上看，该字段位于刻度 0；所以该字段位于，从 8 开始，偏移量为 0 字节的位置，即位于 0x08 字节。

有些字段，如源 IP 字段，长度为多个字节，我们在图中看到，位于 0x12:4。其他一些字段只占用了半字节，如 0x00 字节的高位字节为版本字段，低位字节为 IP 头长度字段。0x06 字节的粒度更细，每一位都表示一个字段。当字段的值为单个二进制数值时，它通常是一个标记（flag）。如 IPv4 头中的翻转（Reversed）、不分片（Don’t Fragment）和多个分片（More Fragments）字段。标记的值为一个一位的二进制数值，1（true）或 0（false），所以当标记值为 1，表示标记生效。标记生效的具体含义根据协议和字段而定。

让我们在图 B-4 中查看另一个例子（这个结构图在第 6 章出现过）。

<table>
<tr><td colspan="8">传输控制协议(TCP)</td></tr>
<tr><td>偏移位</td><td>八位组</td><td colspan="2">0</td><td colspan="2">1</td><td>2</td><td>3</td></tr>
<tr><td>八位组</td><td>位</td><td>0–3</td><td>4–7</td><td colspan="2">8–15</td><td>16–23</td><td>24–31</td></tr>
<tr><td>0</td><td>0</td><td colspan="3">源端口</td><td colspan="3">目标端口</td></tr>
<tr><td>4</td><td>32</td><td colspan="7">序号</td></tr>
<tr><td>8</td><td>64</td><td colspan="7">确认号</td></tr>
<tr><td>12</td><td>96</td><td>Data Offset</td><td>Reserved</td><td colspan="2">标志</td><td colspan="3">窗口大小</td></tr>
<tr><td>16</td><td>128</td><td colspan="3">校验和</td><td colspan="3">紧急指针</td></tr>
<tr><td>20+</td><td>160+</td><td colspan="7">选项</td></tr>
</table>

图 B-4　TCP 的数据包结构图

这个图片展示了 TCP 协议头。根据这张图，我们能够在不知道 TCP 用途的情况下，回答很多与 TCP 数据包有关的问题。假设一个 TCP 数据包的协议头如下方的十六进制数据所示：

```
0646 0050 7c23 5ab7 0000 0000 8002 2000
0b30 0000 0204 05b4 0103 0302 0101 0402
```

使用包结构图，我们能够定位和解析特定的字段。如我们能够发现以下

异步社区Kensuke(18319066421) 专享 请尊重版权

信息。

- 源端口号位于 0x00:2 字节（0x00 至 0x01），十六进制值为 0646（十进制：1606）。
- 目的端口号位于 0x02:2 字节（0x02 至 0x03），十六进制值为 0050（十进制：80）。
- 数据偏移量字段表示协议头长度，位于 0x12 字节的高位字节，十六进制值为 8。

让我们将这些知识用来分析这个神秘的数据包。

B.3 分析一个神秘的数据包

在图 B-2 中，我们看到了一个被部分解析的数据包。我们通过被解析的部分数据来确定，这是一个 TCP/IP 数据包，该数据包在同一网络内的两个设备间传输，然而除此之外，我们对该数据的其他信息并不了解。以下是这个数据包的完整十六进制数据：

```
4500 0034 8bfd 4000 8006 1068 c0a8 6e83
c0a8 6e8a 081a 01f6 41d2 eac6 e115 3ace
5018 fcc6 0032 0000 00d1 0000 0006 0103
0001 0001
```

数据包大小为 52 字节。IP 协议的数据包结构图告诉我们，IP 协议头的标准长度为 20 字节；根据 0x00 字节的低位字节表示的头文件长度，我们确认了这一信息。根据 TCP 的包结构图，我们同样了解到，在没有附加选项的情况下，TCP 协议头的长度也是 20 字节（此处没有列出 TCP 结构图，但是我们在第 6 章中对 TCP 选项进行了深入讨论）。这意味着，数据包的前 40 个字节是 TCP 和 IP 协议头数据，这些数据已经被解析。现在，还剩下 12 字节未被解析。

```
00d1 0000 0006 0103 0001 0001
```

如果没有分析数据包结构的知识，你现在可能会一筹莫展，但是你已经知道了如何将数据包结构图应用于未解析数据。在本例中，已经解析的 TCP 数据包表明，数据的目的端口号是 502。在识别未解析数据时，查看通信使用的端口并不一定会奏效，但这是一个好的切入点。Google 搜索结果显示，502 端口是基于 TCP 的 Modbus 协议的常用端口，该协议用于工业控制系统（ICS）网络。我们将十六进制包数据与 Modbus 数据包结构图进行比较，来核实和分析本例的数据包，如图 B-5 所示。

异步社区Kensuke(18319066421) 专享 请尊重版权

基于TCP的Modbus					
偏移位	八位组	0	1	2	3
八位组	位	0–7	8–15	16–23	24–31
0	0	事件标识		协议标识	
4	32	长度		单元标识	功能码
8+	64+	编码			

图 B-5　基于 TCP 的 Modbus 的数据包结构图

这个数据包结构图根据 Modbus 的应用指导文档制作。图 B-5 结构图表明，位于 0x04:2（相对头部起始位置的偏移量）的长度字段包含一个 7 字节的头部。按照这个偏移量，我们在对应的位置上发现其十六进制值为 0006（对应的十进制值为 6），这表明，紧随这个字段之后有 6 字节，实际也是如此。看起来这的确是基于 TCP 的 Modbus 数据。

将完整的十六进制数据和 Modbus 结构图比较，以下是提取出的信息。

- 事件标识字段位于 0x00:2 字节（0x00～0x01），十六进制值为 00d1。该字段用于将应答和请求进行配对。
- 协议标识字段位于 0x02:2 字节（0x02～0x03），十六进制值为 0000。这表明协议为 Modbus。
- 长度字段位于 0x04:2 字节（0x04～0x05），十六进制值为 0006。这定义了数据部分长度。
- 单元标识字段位于 0x06 字节，十六进制值为 01。表示此数据包用于系统内路由。
- 功能码字段位于 0x07 字段，十六进制值为 03。这表示调用读取保持寄存器（Read Holding Registers）功能，用于从一个系统读取一个数值。
- 根据功能码 3，需要再输入两个数据字段。在 0x08:4 字节发现了参考编号（Reference Number）和单词计数（Word Count），这两个字段的十六进制值均为 0001。

这个神秘的数据包能够按照 Modbus 协议的标准被完全解读。如果你正在对产生这个数据包的系统进行故障处理，以上解析出来的内容应该是你向前推进所需要的信息。就算你不会遇到 Modbus 数据，对于如何使用包结构图处理一个未知的协议和未被解析的数据包，这也是一个很好例子。

异步社区Kensuke(18319066421) 专享 请尊重版权

了解你正在分析的数据的抽象方式，总是一种最好的分析思路。这能帮助你做出更合理和明智的决定，使你能在更多样化的场景中处理数据包。在很多情景下，我只能使用命令行工具，如 Tcpdump，进行数据包分析。因为大部分这样的工具缺少很多应用层协议的分析器，所以手动对数据包中的原始数据进行分析的能力极其重要。

注意

我的一位同事曾经负责在一个高安全级别的环境中进行应急响应。他很清楚他需要检查所负责系统的数据，但是不能接入存储这些数据的特定系统。在他们的工作时间内，他们能做的只有将数据包从特定的会话中打印出来。幸亏掌握了数据包组成和数据包结构分析的基础知识，他能够从打印出的数据中获得他需要的信息。当然，这个分析过程相当缓慢。这是一个极端的场景，但是，这是能证明通用的、与工具无关的知识很重要的例子。

由于以上陈述的原因，花费时间将数据包拆分，以获得使用多种方式分析数据包的经验，对我们很有帮助。我在这方面下了很多功夫：我打印了一些常用协议的数据包结构图并进行封塑，然后将这些图片放在书桌旁。我还在笔记本电脑和平板中保存了电子版本，以便外出时快速查阅。为了方便获取，我在随本书发放的抓包文件的压缩包中包含了一些常用的数据包结构图。

B.4 小结

在本附录中，我们学习了如何解析多种格式的数据包，以及如何使用数据包结构图对未解析的数据进行分析。利用这些知识，你应该能够理解如何在不依赖于特定工具的情况下查看并分析数据包。

异步社区Kensuke(18319066421) 专享 请尊重版权



借助 Wireshark 这款世界流行的网络嗅探器，读者可以很容易地捕获到网络中的数据包，而不管是有线网络还是无线网络。但是，如何使用这些数据包来理解网络状况呢?

本书内容针对 Wireshark 2.x 版本进行了更新，以帮助读者掌握数据包捕获的方法，从而更好地解决网络问题。本书新增了 IPv6 和 SMTP 的相关知识，并讨论了 TShark 和 Tcpdump 两种命令行包分析工具的使用方法。此外，本书还介绍了使用数据包结构图查看和表示数据包的内容。

您将学到：

- 实时监控网络，并参与实时网络通信；
- 构建自定义的捕获过滤器和显示过滤器；
- 使用包分析来解决常见的网络问题，如连接丢失、DNS问题和网络性能缓慢等；
- 在数据包级别探索现代漏洞和恶意软件；
- 从捕获的数据包中提取通过网络发送的文件；
- 绘制流量模式使网络中传输的数据可视化；
- 使用Wireshark高级特性来理解令人困惑的捕获文件；
- 构建统计数据和报告，以便更好地向非技术人员解释技术性的网络信息。

无论是初学者还是有一定经验的网络技术人员，都将通过本书学会如何使用 Wireshark 完成网络任务，从而更加深 入地理解网络概念。

Chris Sanders身兼计算机安全咨询人员、研究人员和教育家等多种角色。他是 *Applied Network Security Monitoring* 的作者并定期在个人的博客上发表文章。

异步社区 www.epubit.com
新浪微博 @人邮异步社区
投稿/反馈邮箱 contact@epubit.com.cn

ISBN 978-7-115-49431-3

分类建议：计算机／网络技术／网络安全
人民邮电出版社网址：www.ptpress.com.cn

异步社区Kensuke(18319066421) 专享 请尊重版权