## Computer Science (Episode 27)

Episode 27 (Security VI: Contraption) Packet Analysis II



Using WireShark

Tanuki, the next step is packet analysis using WireShark. The subject for analysis is the chat program created in episodes 6 and 7. We will use Wireshark to analyze the packets during communication between the chat in the server configuration and the chat in the client configuration.

The ASCII code will also be needed, so I will leave it as presented again.

adecimal																
haracter		1	1	1	1	1	1	1	1	1	1	1	1	1		
16 進	70	71	72	73	74	75	76	77	78	79	7a	7b	7c	7d	70	10
10 進	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	120
文 字	р	q	r	s	t	u	v	w	x	у	z	{	I.	}	~	
16 進	60	61	62	63	64	65	66	67	68	69	6a	6b	6c	6d	60	00
10 進	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	110
文 字	•	a	b	с	d	е	f	g	h	i	j	k	1	m	n	11
10 16 進 進	80 50	81 51	$82\ 52$	83 53	84 54	85 55	$86\ 56$	87 57	$88\ 58$	89 59	90 5a	91 5b	92 5c	93 5d	94 5e	
文 字	Р	Q	R	$\mathbf{S}$	Т	U	V	W	Х	Y	$\mathbf{Z}$	]	¥	]	^	
10 16 進 進	64 40	$65\ 41$	$66\ 42$	67 43	$68\ 44$	$69\ 45$	70 46	$71 \ 47$	$72\ 48$	$73\ 49$	74 4a	75 4b	76 4c	77 4d	78 4e	
文 字	@	А	В	С	D	Е	F	G	Η	Ι	J	Κ	$\mathbf{L}$	Μ	Ν	<u>اا</u>
)16 重進	3 30	31	) 32	L 33	2 34	3 35	4 36	5 37	3 38	7 39	3 3a	) 3b	) 3c	3d	2 3e	
て 10 <b>2</b> 進	48	49	50	51	52	53	54	55	56	57	58	59	: 60	: 61	• 62	
	)	. 1	2		4	Ę	6	7	8 8	9	:	;	<	L =	>	
10 16 進 進	$32\ 20$	$33\ 21$	$34\ 22$	$35\ 23$	$36\ 24$	$37\ 25$	$38\ 26$	$39\ 27$	40 28	$41\ 29$	42 2a	43 2b	44 2c	45 2d	46 2e	
文 字	$\mathbf{SP}$	!	"	#	\$	%	&	'	(	)	*	+	,	-		
16 進	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	
10 進	16	17	18	19	20	21	22	23	<b>24</b>	25	26	27	28	29	30	
文 字	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ЕТВ	CAN	EM	SUB	ESC	$\mathbf{FS}$	GS	$\mathbf{RS}$	
16 進	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	
10 進	0	1	2	3	4	<b>5</b>	6	7	8	9	10	11	12	13	14	
文 字	NUL	SOH	STX	ETX	ЕОТ	ENQ	ACK	BEL	BS	ΗT	LF*	VT	FF*	CR	$\mathbf{SO}$	

From the IT Dictionary

You have studied tcpdump, so you know that when data flows over a communication line, it is divided into packets and sent. Here's a refresher. A packet is made up of data, a header in the upper application layer (e.g., HTTP), a header in the transport layer (e.g., TCP), a header in the network layer (e.g., IP), and a header in the individual network layer (e.g., Ethernet).

In the figure, the following is shown.

←	as	pect of a packe	t flowing through a packet	
Ethernet header	IP header	TCP header	Part of data	FCS
14 bytes	20 bytes	20 bytes		4 bytes
		(UDP:8 byte	s)	trailer

**\*\***FCS(trailer (vehicle)): Checks to see if packets were not corrupted during transmission.

When packets flow from PC (A) to PC (B), capturing those packets is called packet capture, right? Also, looking inside the captured packets is called packet monitoring. Needless to say, seeing and analyzing the actual packets flowing through the system is more useful than learning from a book. But if you don't learn how to operate and view the packet monitoring, it's like a cat with a panda!

As I said before, WireShark is installed on "kali Linux", so you can use it right away.

What if WireShark is not installed on anything other than "kali Linux"? For example, what if I want to use it on CentOS7 which I have built?



It is possible to install on CentOS7, but it is quite a difficult task. It would be faster to install "kali Linux", but just in case, here are the steps to install it on CentOS7.

#### [Installing Wireshark]

 First things fi	rst: Preparation
su - root	# become root
# mkdir tool	# Create a directory tool as root for storage

 2 Next, you need Openssl Make sure Openssl is installed on CentOS.
 # openssl version

If installed, do the following just in case

# yum clean all

- # yum list updates
- # yum update openss1

(If you are installing Openssl anew)

**Dependency Installation** # yum install -y zlib-devel perl-core make gcc Download openssl-1.1.1 source # curl https://www.openssl.org/source/openssl-1.1.1.tar.gz -o /usr/local/src/openss1-1.1.1.tar.gz Installing openssl-1.1.1 # cd /usr/local/src # tar xvzf openss1-1.1.1.tar.gz # openss I-1.1.1/ # ./config --prefix=/usr/local/openss1-1.1.1 shared zlib # make depend # make # make test # make install **Installation Confirmation** # Is - I /usr/local/openss1-1.1.1 operation check # /usr/local/openssl-1.1.1/bin/openssl ciphers -v TLSv1.3 Return to Root's current directory. # cd ~

3Whether Openssl is already installed or a new installation, there are common tasks that must be performed.

# yum install openssl-devel
# yum install libgcrypt-devel

④ python3 is required# yum install python3

⑤ cmake is required Download Site https://cmake.org/download/ The site displays the following

Platform	Files
Unix/Linux Source (has \n line feeds)	cmake-3.16.3.tar.gz
Windows Source (has \r\n line feeds)	cmake-3.16.3.zip
inary distributions:	

#### Save the file as /root/tool/cmake-3.16.3.tar.gz

# cd tool
defrosting
# tar xvfz ./cmake-3.15.3.tar.gz
Go to the unzipped directory
# cd ./cmake-3.15.3
Install required packages
# yum install -y gcc gcc-c++
Create makefile in bootstrap
# ./bootstrap
make & make install
# make
# make install
confirmation
# cmakeversion



Why is cmake needed to install WireShark?



Since WireShark is obtained from a source file, it must be built (including compilation). For this, cmake is necessary.

Now it's time to download and install WireShark.

⑥ Installing WireShark Download Site

https://www.wireshark.org/download.html

# WIRESHARK

## Download Wireshark

The current stable release of Wireshark is 3.2.1.

```
Stable Release (3.2.1)
```

Windows Installer (64-bit)

Windows Installer (32-bit) Windows PortableApps<sup>®</sup> (32-bit)

macOS 10.12 and later Intel 64-bit .dmg

– click

Source Code

Save the file as /root/tool/wireshark-3.2.1.tar.xz

```
# cd tool
defrosting
# tar xJfv ./wireshark-3.0.5.tar.xz
Create a directory for build and enter
# mkdir build
# cd build
Install required packages
# yum install -y gcc gcc-c++ glib2-devel libgcrypt-devel flex-devel
byacc libpcap-devel qt5-qtbase-devel qt5-linguist qt5-qtmultimedia
-devel qt5-qtsvg-devel
Execution of cmake
# cmake .../wireshark-3.0.5
make & install
# make
# make install
confirmation
# tshark --version
If TShark (Wireshark) 3.0.5 is displayed, you are done.
```



Now that I can use the WireShark tool, I'll try to do some packet analysis from the web server, as well as how to operate it. But I'll have to start WireShark with administrator privileges (root) as shown below to get the complete packets for analysis.

Clicking on the WireShark icon won't give me administrator privileges!

アーション	root@kali: /home/ka
ファイル 操作 編集 表示 ヘルプ	
<pre>(kali@kali)-[~] -\$ sudo su sudc] kali Password:     (root@kali) -[/home/kali] -# /bin/wireshark ** (wireshark:2084) 11:41:37.555147 '/tmp/runtime-root'</pre>	[GUI WARNING] QStar
7 7	イヤーシャークネットワークアナ
イル( <u>F</u> ) 編集(E) 表示( <u>V</u> ) 移動( <u>G</u> ) キャプチャ	( <u>C</u> ) 分析( <u>A</u> ) 統計( <u>S</u> ) 電話(y
	← ⇒ 誓 주 ±
示フィルタ <ctrl-></ctrl-> を適用	
Wiresharkへようこそ キャプチャ このフィルタを利用: 📕 キャプチャフィ	イルタ を入力
eth0	him
Loopback: lo	
bluetooth-monitor	and and any second s
mou	

I can read it! I see, you start WireShark after it is Rooted at the terminal. So eth0 is the device name of the NIC. That's what you specify. It looks like the packet has already arrived. But it seems to be difficult if I don't get used to the operation.

Kitsune, I will explain the necessary operations for packet analysis, but you should study the various operations of WireShark by yourself by referring to the Internet or books. As I have said before, the purpose of this site is not to teach you how to use the tools and applications. Roger, if you have the ability to read a book, you can understand how to operate it.

In order to analyze packets using WireShark, it is better to use a demo model for easy-to-understand testing, where the contents of the transmission and reception are known in advance. So, we will use the chat programs created in episodes 6 and 7. Using the server chat program and the client chat program is the best choice since the content of the transmission is known in advance and they are intercommunicating with each other.

First, let's analyze the packets using the chat program. The chat program is running on a single PC.

client process server process IP: 192.168.0.31 IP: 192.168.0.31 Г ~]\$./cctest Г ~]\$./sctest Go ahead! wait Your turn.: aaaaa aaaaa Waiting! Your turn.: bbbbbb Waiting! bbbbbb ccccc Your turn.: CCCCC Your turn.: ddddd Waiting! Waiting! ddddd quit Your turn.: quit Your turn.: Waiting! quit Waiting! quit

[Execution state of the chat program for packet analysis.]



The packet analysis by WireShark is shown below. It is important to understand the meaning of the "a," "b," and "c" parts. If you can't do that, hacking is a dream come true. Please try your best.

#### [Basic Packet Analysis with Wireshark]

* アプリケーション 場所 Wireshark			🚨 _A 日曜日 10:07
	capture3		_ 1
ファイル(F) 編集(E) 表示(V) 移動(G) キャプチャ(C) 分	忻(A) 統計(S) 電話(y) 無線(W) ツール(T) ヘルプ(H)		
	» % k » 🔲 🗖 e o q 🎛		
■ 表示フィルタ <ctrl-></ctrl-> を適用			
No. Time Source	Destination	Protocol	Length Info
_ 1 0.0000000 192.168.0.31	192.168.0.31	TCP	72 50000 → 54896 [PSH,
2 0.0000940 192.168.0.31	192.168.0.31	TCP	66 54896 → 50000 [ACK]
3 8.1746709 192.168.0.31	192.168.0.31	TCP	73 54896 → 50000 [PSH,
4 8.1747622 192.168.0.31 <b>a</b>	192.168.0.31	TCP	66 50000 → 54896 [ACK]
5 14.294768 192.168.0.31	192.168.0.31	TCP	72 50000 → 54896 [PSH.
<ul> <li>Ethernet II, Src: 00:00:00_00:00:00 (00:00:00</li> <li>Internet Protocol Version 4, Src: 192.168.0.3</li> <li>Transmission Control Protocol, Src Port: 5000</li> <li>Data (6 bytes)</li> </ul>	:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:0 1, Dst: 192.168.0.31 0, Dst Port: 54896, Seq: 1, Ack: 1, Len: 6	0:00)	b
	10 45 00 ·····E· f c0 a8 ···c@·@······E·	_	
0020 00 1f c3 50 d6 70 46 bf d2 35 c2 1f 75 2	a 80 18 ···P·pF··5··u*··		
0030         01         56         81         bb         00         00         01         01         08         0a         00         19         75         6           0040         27         56         61         61         61         61         61         0a	a 00 19 V V.aaaaa.		

The above is divided into three parts: [a], [b], and [c].

[a] : One line represents one packet.

Order in which packets are sent: Elapsed time (initially 0): Sending IP Address receiving IP address

Image: No.         Time         Source           1 0.0000000         192.168.0.31         192.168.0.31           2 0.0000940         192.168.0.31         192.168.0.31           rotocol used: Frame length (bytes): Sending port (50000) to receiving port (50000)         1000000000000000000000000000000000000	81 81
1 0.0000000       192.168.0.31         2 0.0000940       192.168.0.31         rotocol used: Frame length (bytes): Sending port (50000) to receiving port (50000)	1 31 ; port (
2 0.0000940 192.168.0.31 Potocol used: Frame length (bytes): Sending port (50000) to receiving p	31 ; port (
otocol used: Frame length (bytes): Sending port (50000) to receiving j	: port (
	-
Protocol Length Info Communication Contro	ol Flags
TCP         72 50000 → 54896 [PSH,         PSH (push): Prepare to	to send
TCP         66 54896 → 50000 [ACK]         ACK: Response confirm	rmatior

From top to bottom.

- •Ethernet header
- •IPv4 header
- •TCP header
- $\cdot$  Order of transmitted data. The entire packet (Ethernet frame) is 72 bytes.



[c]:A specific hexadecimal and ASCII representation of the contents of a single packet

Tanuki, you can't learn analysis just by looking at what is displayed. I will give you the following exercises to try. Practice is an important part of learning. However, repeating the same thing over and over again to avoid mistakes is a waste of time. So studying for the exam is doing a waste of time. It makes sense if you want to become a bureaucrat who can blame mistakes. However, it is important to practice a few times not to prevent mistakes, but to deepen understanding. If you deny this, you're not learning.



I understand. I've always wanted to try it. But I'd also like to check the answers, so please provide me with the answers.

## [Exercise 1] Write the corresponding hexadecimal number from [c] in the following header. Ethernet header (1 4 byte)

Destination MAC address (6 byte)	Sender MAC address (6 byte)	Type (2 byte)

IP header (20 byte)

version (4)	header length (4)	DSCP(6)	ECN (2)		packet length (16)			
	identifier (10	3)		0(1)	F(1)	M(1)	fragment offset (13)	
live time (8) protocol number (8)		nber (8)		$\mathbf{h} \mathbf{\epsilon}$	eader ch	necksum (16)		
	start	ing IP add	ress 192.16	8.0.31	( <b>32</b> ビッ	· <b>b</b> )		
	en	d IP addres	ss 192.168.	0.31(32	2 t yh)			

TCP header (2 0 byte)

starting p	port number 50000	)(16 bit)	endpoint port number 54896(16 bit)
	S	equence number	(32 bit)
	ackr	lowledgment nur	nber (32 bit)
data offset (4)	reserved bit (3)	control flag (9)	window size (16)
	checksum (16 bit)		urgent pointer (16 bit)
TCP option (121	byte)		

option	

DATA (6 byte)

а	а	а	а	а	nl (LF) new line



## [Answers to Exercise 1]

Ethernet hea	der (14byte)							
Destination	/te)	Sender MAC address (6 byte)				Type (2 byte)		
(	00 00 00 00 00 00		00	00 00	00 00 0	00		08 00
IP header (2	2 0 <b>byte</b> )							
version (4)	header length (4)	DSCP(6)	ECN (2)			packet ]	leng	th (16)
4	5	0	0			0	0 3a	
	identifier (10	6)		0(1)	F(1)	M(1)	fra	agment offset (13)
	F5 63				4			0 00
live	time (8) I	protocol nu	umber (8)	header checksum (16)				
	40	06	;	c3 cb				
	star	ting IP add	dress 192.16	8.0.31	( <b>32</b> ビッ	· <b>h</b> )		
			C0 a8 00 1f	•				
	en	d IP addre	ess 192.168.	0.31(32	2 ビット)			
			C0 a8 00 1f	•				
TCP header	(20byte)							
star	ting port number 50	<b>)000(16 bi</b> t	t)	endj	point p	ort nun	nber	54896(16 bit)
	C3 50				D6 70			
		sequenc	e number	(32 bit	)			
			46 bf d2 35					
					>			

	acknowledgment number (32 bit)									
	C2 1f 75 2a									
data offset (4)	data offset (4)reserved bit (3)control flag (9)window size (16)									
8	0 1	8	01 56							
(	checksum (16 bit)		urgent pointer (16 bit)							
	81 bb		00 00							

TCP option (1 2 byte)

option	
01 01 08 0a 00 19 75 0a 00 19 27 56	

## DATA (6 byte)

a	а	а	а	а	nl (LF) new line
61	61	61	61	61	0a



#### [Detailed explanations for packet analysis.]

Ethernet Header Terms

Type: If the upper layer is IP, 0x0800. For ARP (Address Resolution Protocol: MAC address query from broadcast address such as ping, dns, nslookup, etc.), 0x0806.

#### IP Header Terms

Version: 4 for IPv4.

Header length: 4 byte units, so 4 x ? = 20 (bytes) from ? = 5, which is 5.

DSCP, ECN: Indicates the state of packet congestion on the transmission line.

Packet Length: The length (in bytes) of an IP packet. The following calculation formula is used.

Packet length = Length of entire packet (72 bytes) - Ethernet header (14 bytes) = 58 = x003a ldentifier: Increase by I for each packet sent out.

- Flags: O (unused: 0), F (divisible: 0, not divisible: 1), M (last fragment (not fragmented): 0, fragment in progress: 1)
- Fragment Offset: Fragmentation (packet splitting) occurs when a single IP packet exceeds 1500 bytes. Indicates where the fragmented data will be in the original packet; 0x4000 indicates that fragmentation is not possible and no fragmentation has occurred.
- Survival Period: Indicates the number of routers an IP packet can pass through. Each time it passes through a router, the number is decreased by 1.

Protocol number: Indicates the upper protocol: 6 for TCP, 17 for UDP, 1 for ICMP, and 4 for IP. Header checksum: guarantees that the IP header is not corrupted. I's complement of the IP header.

#### TCP Header Terms

Sequence number: Order control, restoring the sent TCP packets in the correct order.

Acknowledgement number: Sequence number + data size received

Data offset: TCP header length (5 for 20 bytes). In this case, 32 bytes including the TCP option, so 8 is entered as 32/4.

Reserved bits: 3 bits reserved for future expansion. Currently unused and set to 0.

- Control flags: I bit flags (9 bits in total) for connection related control (NS, CWR, ECE, URG, ACK, PSH, RST, SYN, FIN).
- Window size: Notifies the receiver of the size of the data to be sent and prepares the receive buffer. In this case, 0x0156 is used to notify the transmission of 342 (decimal) bytes of data.
- Checksum: Ensures that TCP packets are not corrupted.

Urgent pointer: Indicates the location and number of bytes of data that must be processed urgently.

Connectionless UDP does not include the sequence number, acknowledgment number, or control flags required for a connection.



Kitsune, the packets sent and received are flying in and out at a dizzying rate, is there any way to save them to a file and analyze them carefully?

Yes, there is.

If you execute the following command with Root privileges, the file (cap1) will be saved. However, when you open the cap1 file, you must start WireShark and open it as a file for WireShark. At this time, you do not need to have Root privileges to open the file.

]#tshark -i eth0 -w /home/cap1

(Note) The cap1 file created in /home/ does not have access rights, so grant access rights as follows.

]# chmod 777 /home/cap1



I understand that [Detailed Explanation for Packet Analysis] is an important term, but it's not easy to understand. Do I have to learn this?

You don't have to memorize it, but I'd like you to understand how it works as you do your practice assignments. Next, I should try to analyze the packets from the web server. There is a site that provides a demo page for analysis, so let's use that. The site is not using OpenSSL, so it is accessible via http, not https, and the HTTP header and data portions are clearly distinguishable. The following is the result of accessing this site and capturing the data with WireShark. The IP address of "www.ikeriri.ne.jp" is 163.44.9.71. The IP address of the "kali Linux" that is capturing the packets is 192.168.0.29.

Sample × +
 ← → C △ ○ △ www.ikeriri.ne.jp/sample.html
 Kali Linux ♣ Kali Tools ▲ Kali Docs ★ Kali Forums < Kali NetHu</li>
 homepage

4		*eth0		
ファイル( <u>F</u> ) 編集( <u>E</u> )	表示(⊻) 移動( <u>G</u> ) キャプチ	ャ( <u>C</u> ) 分析( <u>A</u> ) 統計( <u>S</u> )	電話(ỵ) 無線( <u>W</u> ) ツール	レ(I) ヘルプ(H)
	📄 🛅 🖹 🎑 ۹	🔶 🚔 🐳	🛃 📃 🔍 🤄	2 @ 1
tcp.port == 80				×
No. Time 1239 30.2716253 1240 30.2793710 1241 30.2880457 1242 30.2880777 1263 33.171500 1265 33.171500 1266 36.2966540 Transmission Cor Hypertext Transf - Line-based text htm<br <html>\n <head>\n <title>sample </title></head>\n <body>\n <h1>h00000000000000000000000000000000000</h1></body></html>	Source Source	Destination 163.44.9.71 192.168.0.29 192.168.0.29 163.44.9.71 192.168.0.29 163.44.9.71 192.168.0.29 163.44.9.71 192.168.0.29 163.44.9.71 192.168.0.29 163.44.9.71 192.168.0.29 163.44.9.71 192.168.0.44 5)	Protocol         Length         Info           HTTP         431         GET         /           TCP         66         80 →         +           HTTP         356         HTTP/         TCP         66         49148           HTTP         431         GET         /         +	<pre>'sample.html HTTP/1.1 49148 [ACK] Seq=1 Ack=366 1.1 200 0K (text/html)</pre>
\n	· · · · · · · · · · · · · · · · · · ·			
0000         76         65         0d         0a           00e0         3a         20         74         65           00f0         72         73         65         74           0100         64         6f         63         74           0110         74         6d         6c         3e           0120         6c         65         3e         73           0130         3e         0a         3c         2f           0140         0a         3c         68         31           0150         31         3e         0a         3c           0160         6c         3e         0a         3c	43       6f       6e       74       65       6e       74       20         78       74       2f       68       74       6d       6c       31         3d       55       54       46       2d       38       0d       0a         79       70       65       20       68       74       6d       6d       0a         79       70       65       20       68       74       6d       6d       0a         61       6d       70       6c       65       3c       2f       74       6d       6d       65         61       6d       70       6c       65       3c       2f       74       6d       6d       62         61       6d       70       6c       65       3c       2f       62       61       64       3e       0a       3c       62         68       6f       6d       65       70       61       67       2f       62       6f       64       79       3e       0a       3d         2f       62       6f       64       79       3e       0a       3d       3d       3d	1       54       79       70       65       ve       ve         0       20       63       68       61       : te:         a       0d       0a       3c       21       rset:         c       3e       0a       3c       21       rset:         c       3e       0a       3c       21       rset:         c       3e       0a       3c       68       doct'         a       3c       74       69       74       tml>         4       69       74       6c       65       le>s:       2         2       6f       64       79       3e       -<	Cont ent-Type xt/h tml; cha =UTF -8···· <br ype html>· <h add&gt;·<tit ampl eHead &gt;-bod epage<td></td></tit </h 	
🔵 🍸 Line-based t	ext data (data-text-lines), 10	リコバイト パケ・	ット数: 1268 · 表示: 23 (1.8%	6)・欠落:0(0.0%) プロファイル

Tanuki, as [Exercise 2], start wireshark and also start a browser. Access the "http://www.ikeriri.ne.jp/sample.html" site from your PC. Extract the first and last lines of the http header section from (b) of wireshark at that time. If you are asked to look at part (c) and count the number of bytes in the http header, will you be able to practice and answer the question?

[Answers to Exercise 2]

First line : GET / HTTP/1.1 Last line : Connection: Keep-Alive Bytes: Count them yourself!

> Let's continue. Start wireshark and a browser as "Exercise 3". Extract from (b) the line that borders the http header part and the data part when the data is sent from the "http://www.ikeriri.ne.jp/sample.html" site to the PC. Also, count the number of bytes in the http header section in (c). If you were asked to do this, would you be able to answer the question?

[Answers to Exercise 3]

boundary line : Content-Type: text/html or blank (¥r¥n: carriage return/line feed) Bytes: Count them yourself!

Now, when I capture https using WireShark, I see OCSP protocol packets. What is OCSP? It is a packet that queries an external OCSP responder to see if the digital certificate sent by the site you are accessing is correct. It is shown in the next figure.



So [exercise 4].

Launch WireShark, then launch your browser and visit the "https://www.yahoo.co.jp" site.

Answer the encryption name used in the OCSP protocol hash algorithm displayed at that time, the number of bytes of the issuerNameHash (issuer name), the number of bytes of the issuerKeyHash (issuer's public key) and the serial number.

[Answers to Exercise 4] OCSP Protocol

Hash algorithm : SHA-1 issuerNameHash(Publisher Name) : 20 byte issuerKeyHash(Public key of the issuer) : 20 byte serial number : 16 byte

Next is the UDP protocol.

Capturing the DNS protocol using WireShark shows that the UDP protocol is used. So [exercise 5].

Looking at the displayed UDP header, can you fill in the following table with the header contents in hexadecimal?

#### UDP header

Starting port number(2byte)	Endpoint port number(2byte)
Packet length(2byte)	Checksum(2byte)

#### [Answers to Exercise 5] UDP header

Starting port number (2 byte)	Endpoint port number (2 byte)
A4 7d	00 35 (53:DNS Server)
Packet length (2 byte)	Checksum (2 byte)
00 2b (43 byte)	60 e8

[Exercise 6] and it's the last one.

Launch WireShark (specify DNS protocol), then launch your browser and read the following flags from the DNS header that appears when you access the "https://www.yahoo.co.jp" site . First, how many bytes are in the DNS header? One thing to note, however. There are two types of DNS headers, one in the request packet and the other in the response packet. Both have the same format shown in the table below, but the values are different.

#### [Requests.]

	I D (16 bit)										
Express	Expressed in hexadecimal :										
QR(1)	Opcode(4)	AA(1)	TC(1)	RD(1)	RA(1)	Z(1)	AD(1)	CD(1)	RCODE(4)		
	binary digits :										
Respon	se.]										

				ID (1	6 bit)				
Expres	sed in hexadecim	nal :							
QR(1)	Opcode(4)	AA(1)	TC(1)	RD(1)	RA(1)	Z(1)	AD(1)	CD(1)	RCODE(4)
	binary digits :								

#### [Answers to Exercise 6] DNS Header

[Requests.]

I D (16 bit)										
Expressed in hexadecimal : 5d 30										
QR(1)	QR(1) Opcode(4) AA(1) TC(1) RD(1) RA(1) Z(1) AD(1) CD(1) RCODE(4)									
0	0         binary : 0000         0         0         1         0         0         0         0         0000									
	01 00 (hexadecimal)									

[Response.]

	I D (16 bit)											
Expressed in hexadecimal : 5d 30												
QR(1)	QR(1) Opcode(4) AA(1) TC(1) RD(1) RA(1) Z(1) AD(1) CD(1) RCODE(4)											
1	1         binary : 0000         0         0         1         1         0         0         0         0000											
	81 80 (hexadecimal)											



Finally, an encore to the exercise. Consider what can be read from the above table from the bit sequence of flags, divided into [request] and [response].

[Answers to Encore for Practice] DNS Header

[request] : QR=0 indicates a query.Opcode=0 indicates a normal query; RD=1
indicates a full-service resolver.
[response] : QR=1, response. From Opcode=0, normal query. From RD=1, full
service resolver. From RA=1, it is clear that name resolution is
possible.



I'll give you a detailed supplementary explanation of DNS headers, see if you need it.

### [Supplemental DNS header description.]

The DNS header (application layer) is shown in the table below.

I D (16 bit)													
QR(1)	Opcode(4)	AA(1)	TC(1)	RD(1)	RA(1)	Z(1)	AD(1)	CD(1)	RCODE(4)				
	QDCOUNT(16 bit)												
ANCOUNT(16 bit)													
NSCOUNT(16 bit)													
ARCOUNT(16 bit)													

ID: Specified at the time of query (inquiry) and copied at the time of response (reply). QR:Inquiry 0, response 1.

Opcode:Normal query 0, Notify is 4, and Update is 5.

RD:Name resolution. Query authoritative DNS servers0 and full-service resolvers (DNS servers that look at their own cache and ask them to tell you if they don't know)1.

\_\_\_\_\_

RA : Name resolution is possible 1. Z:Future Reservations. Always 0.

:

:

As shown in the figure below, there are two types of DNS headers: packets in the request and packets in the response.Both have the same format shown in the table above, but the values are different.

[reque	st]															
👻 Doma	in	Nam	e S	yst	em	( qı	iery	/)				,				
Transaction ID: 0x5d30																
Flags: 0x0100 Standard query Questione: 1																
4														1122		
0000	00	0d	02	d4	ec	9e	94	de	80	07	c8	c9	08	00	45	00
0010	00	3†	be	c1	40	00	40	11	fa	7b	<u>co</u>	-88	00	1†	сo	a8
0020	00	01	a4	7d	00	35	00	2b	c5	dq	5d	30	01	00	00	01
0030	00	00	00	00	00	00	05	6C	61	67	71	69	05	79	61	68
0040	6†	6†	02	63	61	02	6a	70	00	00	10	00	01			
[Response.]									ID is copied.							
Litespor	196.]						,						5 19	cobi	eu.	
<ul> <li>Doma</li> </ul>	in	Nam	e S	yst	em	(re	esp	onse	e)				7 18	copi	cu.	
- Doma Tr	in ansa	Nam act	e S ion	yst ID	em : 0	(re 0x50	espo 130	onse	e)			11	J 15		eu.	
Doma     Tr     Fl	in ansa ags	Nam act : 0	e S ion x81	yst ID 80	em : 0 Sta	(re x5d inda	espo 130 Ird	onse que	e) ry 1	res	por	se,	No	) er	roi	<b>5</b> 00
Doma     Tr     Fl	in ansa ags	Nam act : 0:	e S ion x81	yst ID 80	em : 0 Sta	(re x5d inda	ispo 130 Ird	onse que	e) ry	res	por	se,	No	) er	roi	<b>6</b> 70
Doma     Tr     Fl:     O000	in ansa ags 94	Nam act : 0: de	e S ion x81 ~ 80	yst ID 80 1	em : 0 Sta c8	(re x5d inda c9	9 130 1rd 00	onse que 0d	e) ry 02	res d4	por ec	se, 9e	Nc 08	o er	-roi 45	10
<ul> <li>✓ Doma</li> <li>Tr</li> <li>▶ F1.</li> <li></li> <li><td>in ansa ags 94 00</td><td>Nam act : 0: de 5c</td><td>e S ion x81  80 43</td><td>yst ID 80 1 07 ee</td><td>em : 0 Sta c8 40</td><td>(re 0x5d inda 00</td><td>9 130 17d 00 40</td><td>onse que 0d 11</td><td>e) ry 02 75</td><td>res d4 22</td><td>por ec</td><td>se, 9e</td><td>Nc 08</td><td>o er 00 01</td><td>-roi 45 c0</td><td>10 a8</td></li></ul>	in ansa ags 94 00	Nam act : 0: de 5c	e S ion x81  80 43	yst ID 80 1 07 ee	em : 0 Sta c8 40	(re 0x5d inda 00	9 130 17d 00 40	onse que 0d 11	e) ry 02 75	res d4 22	por ec	se, 9e	Nc 08	o er 00 01	-roi 45 c0	10 a8
<ul> <li>Doma</li> <li>Tr</li> <li>F1</li> <li>0000</li> <li>0010</li> <li>0020</li> </ul>	in ansa ags 94 00 00	Nam act : 0: de 5c 1f	e S ion x81  80 43 00	yst ID 80 1 07 ee 35	em : 0 Sta c8 40 a4	(re 0x5d inda c9 00 7d	9 130 17d 00 40 00	que que 0d 11 48	e) ry 02 75 0a	res d4 22 2f	por ec	se, 9e	Nc 08 00 31	00 01 80	45 00	10 a8 01
<ul> <li>Doma Tr</li> <li>F1.</li> <li>0000</li> <li>0010</li> <li>0020</li> <li>0030</li> </ul>	in ansa ags 94 00 00	Nam act. 0 0 0 0 1	e S ion x81  80 43 00 00	yst ID 80 1 07 ee 35 00	em : 0 Sta c8 40 a4 00	(re 0x5d 0nda 00 7d 00	9 130 17d 00 40 00 05	onse que 0d 11 48 6c	e) 9ry 02 75 0a 6f	res d4 22 2f 67	por ec	se, 9e	Nc 08 00 31 05	00 01 80 79	45 c0 61	10 a8 01 68
<ul> <li>Doma</li> <li>Tr</li> <li>F1</li> <li>0000</li> <li>0010</li> <li>0020</li> <li>0030</li> <li>0040</li> </ul>	in ansa ags 94 00 00 6f	Nam act : 0: ion de 5c 1f 01 6f	e S ion x81 80 43 00 00 02	yst ID 80 1 07 ee 35 00 63	em : 0 Sta c8 40 a4 00 6f	(re 0x5d 0da 00 7d 00 02	9 sp 130 1rd 00 40 00 05 6a	onse que 0d 11 48 6c 70	e) ry 02 75 0a 6f 00	res d4 22 2f 67	por ec 5d	9e 30 60	NC 08 00 31 05 01	00 01 80 79 c0	45 c0 61 0c	10 a8 01 68 00
<ul> <li>✓ Doma</li> <li>Tr</li> <li>▶ F1.</li> <li></li> <li><td>in ansa ags 94 00 00 6f 05</td><td>Nam act. : 0: de 5c 1f 01 6f 00</td><td>e S ion x81  43 00 00 02 01</td><td>yst ID 80 1 07 ee 35 00 63 00</td><td>em : 0 Sta 28 40 a4 00 6f 00</td><td>(re x5d nda c9 00 7d 00 02 02</td><td>9 sp 130 17d 00 40 05 6a 6f</td><td>onse que 0d 11 48 6c 70 00</td><td>e) ry 02 75 0a 6f 00 11</td><td>res d4 22 2f 67 00</td><td>por ec 5d 1c</td><td>se, 9e 30 60 64</td><td>Nc 08 00 31 05 01 67</td><td>00 01 80 79 c0</td><td>45 c0 61 0c 61</td><td>10 a8 01 68 00 6c</td></li></ul>	in ansa ags 94 00 00 6f 05	Nam act. : 0: de 5c 1f 01 6f 00	e S ion x81  43 00 00 02 01	yst ID 80 1 07 ee 35 00 63 00	em : 0 Sta 28 40 a4 00 6f 00	(re x5d nda c9 00 7d 00 02 02	9 sp 130 17d 00 40 05 6a 6f	onse que 0d 11 48 6c 70 00	e) ry 02 75 0a 6f 00 11	res d4 22 2f 67 00	por ec 5d 1c	se, 9e 30 60 64	Nc 08 00 31 05 01 67	00 01 80 79 c0	45 c0 61 0c 61	10 a8 01 68 00 6c

A DNS packet is formed by a DNS header + data (Question section, Answer section, Authority section, and Additional section).

The data portion also includes a variable-length portion by domain name.



Translated at DeepL