Report on Programming Assignment 3

Gaurang Naik 123079009 Collaborator: Aniruddh Rao

October 7, 2014

1 Experiment Setup

The setup for the assignment is as shown in figure 1. There are two systems connected to each other over the CSE LAN interface. These machines are designated the names C (for Client) and S (for Server) respectively. The IP address of C is 10.129.5.195 and that of S is 10.129.5.194. At each machine, three virtual interfaces are created (using ifconfig eth0:x 192.168.y.x, where y=6 for S and y=7 for C and x=1,2,3). Thus, there are three virtual interfaces at each machine C (C1, C2 and C3) and S (S1, S2 and S3).

With no soultion implemented, the virtual interfaces at C cannot connect to the virtual interfaces at S. This is because C has no idea as to what the private IPs of S mean, and vice versa. The objective of this assignment is to implement a solution at C and at S, so the virtual interfaces at C and S can communicate with each other.



Figure 1: Topology of clients and servers

For testing the communication between the Clients and the Servers at C1, C2, C3 and S1, S2, S3, I have tested the Client Server application implemented in PA1. As mentioned in the assignment description, the client and server were modified to operate on a particular IP interface.

1.1 Solution using iptables

The first solution for providing connectivity between C1, C2, C3 and S1, S2, S3 has been implemented using iptables. The iptables commands are executed at nodes C and S.

The idea behind the iptables implementation is shown in figure 2. The entire solution can be broken down into three steps.



Figure 2: Solution using iptables

- 1. In order that the virtual interfaces of C (C1, C2 and C3) can communicate with the outside network, we first need to perform a Source Network Address Translation (SNAT) at node C. The same thing can be achieved by using MASQUERADE at node C. Once this is done, virtual interfaces of C can communicate with S.
- 2. Now, C has no clue what to do with packets destined to S1, S2 and S3. Therefore, a Destination NAT (DNAT) is implemented at C, which translates all packets destined to S1, S2 and S3 to the gateway S. At this stage, C1, C2 and C3 appear to communicate with S1, S2 and S3. However, they simply talk to the gateway S. Each virtual interface is identified uniquely by using different port numbers while performing DNAT. As shown in figure 2, any packet arriving from 192.168.6.41:5000 will be translated to 10.129.5.194:6001; any packet arriving from 192.168.6.42:5000 will be translated to 10.129.5.194:6002 and so on.
- 3. Now that packets destined to S1, S2 and S3 reach S successfully, a DNAT is implemented at S. Any packet arriving at port 6001 of S is translated to 192.168.6.41:5000, any packet arriving at port 6002 of S is translated to 192.168.6.42:5000 and so on. Thus, packets destined to S1, S2 and S3 arrive at the respective interfaces correctly.

Note that the solution is very rigid. I was unable to make a more generic solution. The servers have to be necessarily listening on port 5000 to make communication possible.

1.2 Solution using tun

The second solution for providing connectivity between C1, C2, C3 and S1, S2, S3 has been implemented using the tun interface. tun interfaces were created nodes C and S.

An overview of the working of this solution is shown in figure 3.



Figure 3: Soultion using tun

The virtual interfaces now do not communicate with the inteface eth0. Instead, *all packets from the virtual interfaces are routed to the tun interface using the route command.* At C and S, a user space application runs which simply exchanges data between the tun interface and the eth0 interface. This application is simply a modification of the sample code provided as reference in the assignment description.

A script file is used to create a persistent tun interface at C and S. A *select()* loop runs in the application which checks if any data arrives on the tun or eth0 interface. The application uses the tun file descriptor and the socket descriptor and to check data arriving on either interface. Any data received from the tun interface is sent to the eth0 interface and vice versa.

Any IP packet sent to the tun interface is captured by the user space application, which sends this IP datagram over a TCP socket. As a result, one IP datagram gets encapsulated into another IP datagram, hence the tunneling. At the receiver side, the user space application reads data from eth0 interface over the TCP socket and sends this data to the tun interface. This is how the decapsulation occurs. The tun interface sends the packet to the appropriate destination virtual interface.

2 Wireshark Analysis of iptables implementation

The Wireshark capture for the iptables implementation at S is as shown in figure 4.

As seen in figure 2, all packets to be sent to 192.168.6.4x are sent to port 600x (x = 1, 2, 3) of S. This can clearly be seen in figure 4. The communication happens between ports 6002 and 6003 (6001 not seen in image) of S and randomly allocated ports of C. This is as a result of iptables DNAT implementation at C. Also, due to the SNAT implementation at C, all packets arriving from C1, C2 and C3 seem to arrive from C (10.129.5.195). Finally, the DNAT implementation at S sends the packet to appropriate interface (S1, S2 or S3).

The highlighted packet in the trace contains a response from the server (of PA1). The contents at the bottom show the response (Ok anirudh).

3 Wireshark Analysis of tun implementation

The Wireshark capture for the tun implementation at S is as shown in figure 5 and 6.

Here, the captures have been taken on the tun interface and also on the eth0 interface. The eth0 interface capture (figure 5) shows that the TCP connection was set up only once between C and S. At the tun interface (figure 6), a TCP connection using SYN, SYN ACK and ACK was set up for each flow (C1-S1, C2-S2, C3-S3).

	i 🗟 🗐 (🌬 l 📔 🖾 🗶 😋	🚊 🔍 🔶 🤿 '	€ Ŧ .	↓ [🍑 📔 法 📀	
Filter:			▼ Expres	sion Cl	ear App	oly			
No.	Time	Source	Destination	Protocol	Length	Source Port	Destination Port	Info	
9	36.579223	20:89:84:a0:e2:45	74:86:7a:42:62:db	ARP	42			Who has 10.129.5.194? Tell 10.129.5.195	
10	36.579645	74:86:7a:42:62:db	20:89:84:a0:e2:45	ARP	60			10.129.5.194 is at 74:86:7a:42:62:db	
11	50.743677	10.129.5.195	10.129.5.194	TCP	74	47318	6003	47318 > x11-3 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 S/	ACK_F
12	50.744076	10.129.5.194	10.129.5.195	TCP	74	6003	47318	x11-3 > 47318 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0	MSS:
13	50.744133	10.129.5.195	10.129.5.194	TCP	66	47318	6003	47318 > x11-3 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSva	1=184
14	52.336395	10.129.5.195	10.129.5.194	TCP	69	47318	6003	47318 > x11-3 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=3	TSva
15	52.336789	10.129.5.194	10.129.5.195	TCP	66	6003	47318	x11-3 > 47318 [ACK] Seq=1 Ack=4 Win=14592 Len=0 TSva	1=182
16	52.336827	10.129.5.194	10.129.5.195	TCP	77	6003	47318	x11-3 > 47318 [PSH, ACK] Seq=1 Ack=4 Win=14592 Len=1	1 TSV
17	52.336901	10.129.5.195	10.129.5.194	TCP	66	47318	6003	47318 > x11-3 [ACK] Seq=4 Ack=12 Win=29312 Len=0 TSva	al=18
18	54.160877	10.129.5.195	10.129.5.194	TCP	72	47318	6003	47318 > x11-3 [PSH, ACK] Seq=4 Ack=12 Win=29312 Len=	5 TSv
19	54.161346	10.129.5.194	10.129.5.195	TCP	77	6003	47318	x11-3 > 47318 [PSH, ACK] Seq=12 Ack=10 Win=14592 Len:	=11 T
20	54.161442	10.129.5.195	10.129.5.194	TCP	66	47318	6003	47318 > x11-3 [ACK] Seq=10 Ack=23 Win=29312 Len=0 TS	/al=1
21	57.161308	10.129.5.195	10.129.5.194	TCP	79	47318	6003	47318 > x11-3 [PSH, ACK] Seq=10 Ack=23 Win=29312 Len:	=13 T
22	57.161789	10.129.5.194	10.129.5.195	TCP	77	6003	47318	x11-3 > 47318 [PSH, ACK] Seq=23 Ack=23 Win=14592 Len	=11 T
23	57.161887	10.129.5.195	10.129.5.194	TCP	66	47318	6003	47318 > x11-3 [ACK] Seq=23 Ack=34 Win=29312 Len=0 TS	/al=1
24	60.617873	10.129.5.195	10.129.5.194	TCP	69	44183	6002	[TCP segment of a reassembled PDU]	
25	60.618268	10.129.5.194	10.129.5.195	ТСР	66	6002	44183	x11-2 > 44183 [ACK] Seq=1 Ack=4 Win=14592 Len=0 TSva	1=182
26	60.618305	10.129.5.194	10.129.5.195	TCP	77	6002	44183	[TCP segment of a reassembled PDU]	
27	60.618379	10.129.5.195	10.129.5.194	TCP	66	44183	6002	44183 > x11-2 [ACK] Seq=4 Ack=12 Win=29312 Len=0 TSva	al=18
28	62.379306	10.129.5.195	10.129.5.194	TCP	72	44183	6002	[TCP segment of a reassembled PDU]	
▶ Frame	19: 77 bytes	s on wire (616 bits).	77 bytes captured (616	bits)					
▶ Ethern	et II. Src:	74:86:7a:42:62:db (74	:86:7a:42:62:db), Dst:	20:89:84	:a0:e2:4	45 (20:89:84	1:a0:e2:45)		
▶ Intern	et Protocol	Version 4, Src: 10.12	9.5.194 (10.129.5.194)	, Dst: 10	.129.5.	195 (10.129	.5.195)		
▶ Transm	ission Contr	rol Protocol, Src Port	: x11-3 (6003), Dst Po	rt: 47318	(47318), Sea: 12,	Ack: 10, Len: 1	1	
▶ Data (11 bytes)								
									_
0000 20	89 84 a0 e	2 45 74 86 7a 42 62 d	b 08 00 45 00E	t. zBb	Ε.				
0010 00	c 3 17 73 b	0 00 40 06 6a e9 0a 8 8 d6 ed 4d 7f 10 ca 9	1 05 C2 0a 81 .r.1@. 5 96 a6 80 18 s	. у м					
0030 00	72 ea 4a 0	0 00 01 01 08 0a 00 1	b d3 98 00 1c .r.J						
0040 3a	bf 4f 4b 2	0 61 6e 69 72 75 64 6	8 0a :.OK a	ni rudh.					
🔵 File: "J	part1-c.pcap"	7911 Bytes 00: Packets	: 60 Displayed: 60 Marked	: 0 Load tin	ne: 0:00.1	117		Profile: Default	

Figure 4: Wireshark capture for iptables (client)

Figure 6 clearly shows communication occuring between the virtual interfaces of C and S. Packets are sent and received by IP addresses 192.168.6.4x and 192.168.7.4x (x = 1, 2, 3).

4 Comparison of two solutions

As mentioned before, the solution using iptables is rigid. The code written allows for communication between the client and server only on port 5000 of the virtaul interfaces. That is, S1, S2 and S3 can hear only on port 5000. And packets sent from C1, C2 and C3 will go through only for those destined to port 5000 of S1, S2 and S3. However, a more generic code could be written for the same purpose making the code more flexible. Also, the solution provided would only work for TCP connections. In order to specify the ports, iptables command require you to sepcify the protocol. In order to use another protocol, say ping, instead of TCP, the protocol needs to be changed to icmp instead of TCP.

As far as the signalling overheads are concerned, when the iptables solution is used, C and S would have to establish a TCP connection for each client-server pair. If there are large number of clients and servers, then this overheads would be significant. On the other hand, using the tun interface, C and S set up a TCP connection only once. TCP connections between the virtual interfaces of C and S are established through the tunneled packets. So all TCP SYN, SYN ACK and ACK packets between say C1 and S1 appear as normal IP packets at C and S eth0 interface. The actual number of TCP connections established in the tun-based solution is one more than the ip-tables based solution (1 TCP connection between C and S, and 1 TCP connection for each virtual client-server pair). As a result, the total number of packets sent and received by C and S is more in the tun based solution than the IP based solution. Thus, if signalling overheads at C and S are concerned, the overheads are larger in iptables-based solution. However, the actual number of packets exchanged between C and S are more in the tun-based solution.

۲	۵ 🦲 🔳	🙍 🚞 🗎 🗴	(C Q ()	€ ₹ 1			🎬 🖻 ங 🔀 🕜	
Filter	: tcp		▼ Exp	ression Clear	Apply Save F	ilter		
No.	Time	Source	Destination	Protocol D	estination Port	Info		
	1 0.000000	10,129,5,194	10.129.5.195	тср	4267	2 60001 > 42672	2 [FIN. ACK] Seg=1 Ack=1 Win=114 Len=0 TSval=2009923	TSecr
	2 0.003370	10.129.5.195	10.129.5.194	TCP	6000	1 42672 > 60001	L [ACK] Seg=1 Ack=2 Win=229 Len=0 TSval=2036330 TSec	r=2009
	5 56.660473	10.129.5.195	10.129.5.194	TCP	6000	1 42673 > 60001	L [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK PERM=1 T	Sval=20
	6 56.660559	10.129.5.194	10.129.5.195	TCP	4267	3 60001 > 42673	3 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SA	CK_PER
	7 56.660915	10.129.5.195	10.129.5.194	TCP	6000	1 42673 > 60001	L [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=2050494 TS	ecr=202
	10 67.712300	10.129.5.195	10.129.5.194	TCP	6000	1 42673 > 60001	L [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=2 TSval=20532	57 TSe
	11 67.712387	10.129.5.194	10.129.5.195	TCP	4267	3 60001 > 42673	8 [ACK] Seq=1 Ack=3 Win=14592 Len=0 TSval=2026851 TS	ecr=20
	12 67.712775	10.129.5.195	10.129.5.194	TCP	6000	1 42673 > 60001	L [PSH, ACK] Seq=3 Ack=1 Win=29312 Len=60 TSval=2053	257 TS
	13 67.712888	10.129.5.194	10.129.5.195	TCP	4267	3 60001 > 42673	3 [ACK] Seq=1 Ack=63 Win=14592 Len=0 TSval=2026851 T	Secr=20
	14 67.713025	10.129.5.194	10.129.5.195	TCP	4267	3 60001 > 42673	3 [PSH, ACK] Seq=1 ACK=63 Win=14592 Len=2 TSVal=2026	851 TS
	15 67.713257	10.129.5.195	10.129.5.194	TCP	6000	1 426/3 > 60001	L [ACK] Seq=63 ACK=3 W1n=29312 Len=0 ISVal=2053257 I	Secr=20
	10 07.713290	10.129.5.194	10.129.5.195	TCP	4207	3 00001 > 420/3	3 [PSH, ACK] Seq=3 ACK=03 WIN=14592 Len=00 ISVal=202	0851 T:
	10 67 712027	10.129.5.195	10.129.5.194	TCP	6000	1 42073 > 00001	L [ACK] SEQ-05 ACK-05 WIN-29512 LEN-0 ISVAL-2053257	2257 1
	19 67 751854	10.129.5.195	10.129.5.194	TCP	4267	3 60001 > 42673	2 [ACK] Seg=63 Ack=65 Win=14502 Len=0 TSval=2026861	TSecr='
	20 67 752240	10.129.5.194	10.129.5.195	TCP	4207	1 42673 > 60001	[[DSH ACK] Seg-65 Ack-63 Win-20312 Len-52 TSval-2020001	53267
	21 67 752351	10.129.5.195	10.129.5.194	TCP	4267	3 60001 > 42673	R [ACK] Seq=63 Ack=117 Win=14592 Len=A TSval=2026861	TSecr:
	22 73 904638	10.129.5.195	10.129.5.194	TCP	6000	1 42673 > 60001	[PSH_ACK] Seg=117 Ack=63 Win=29312 Len=2 TSval=20	54805
	23 73,904699	10.129.5.194	10.129.5.195	TCP	4267	3 60001 > 42673	[ACK] Seg=63 Ack=119 Win=14592 Len=0 TSval=2028399	TSecr
	24 73 065163	10 120 5 105	10 120 5 104	TCD	6000	1 42673 - 60001	[DCH ACK] Con-110 Ack-63 Win-20212 Lon-60 Tous1-2	05/1905
▶ Fran	ne 14: 68 byte:	s on wire (544 bits), 68 bytes captured (5	44 bits)				
► Ethe	ernet II, Src:	Dell_42:62:db (74:	86:7a:42:62:db), Dst: C	ompalIn_a0:e2:4	5 (20:89:84:a0:e	2:45)		
▶ Inte	ernet Protocol	Version 4, Src: 10	.129.5.194 (10.129.5.19	4), Dst: 10.129	.5.195 (10.129.5	.195)		
▶Trar	ismission Cont	rol Protocol, Src P	ort: 60001 (60001), Dst	Port: 42673 (4	2673), Seq: 1, A	ck: 63, Len: 2		
▶Data	a (2 bytes)							
0000	20 80 84 50 4	2 45 74 95 72 42	60 db 00 00 45 00	Et alb E				
0000	20 09 04 d0 0	10 00 40 06 10 9f	02 00 00 00 45 00 0a 81 05 c2 0a 81 6	EL. ZDDE.				
0020	05 c3 ea 61 a	a6 b1 b3 23 1c 02	e9 b6 58 7f 80 18	a#X				
0030	00 72 70 e5 0	00 00 01 01 08 0a	00 le ed 63 00 lf .rp	c				
0040	54 89 00 3c		Т	<				
0 💅	File: "/home/ar	nirudh/cs641-pa3	Packets: 187 · Displayed: 1	83 (97.9%) · Load	time: Profile	: Default		

Figure 5: Wireshark capture for tun (client) at eth0 interface

Filter:			▼ E	xpression Cle	ear App	У					
NO.	Time	Source	Destination	Protocol	Length	Source Port	Destination Port	Info			
3	4 40.42188	200(192.168.7.43	192.168.6.43	TCP	40	56691	5000	56691 > 500	[RST] Seq=	=15 Win=0 Len=0	
3	5 44.41609	900(192.168.7.41	192.168.6.41	IPA	55	51711	5000	unknown 0x0a	[Malformed	d Packet]	
3	6 44.45571	200(192.168.6.41	192.168.7.41	TCP	52	5000	51711	5000 > 5171	[ACK] Seq=	=1 Ack=4 Win=14592 Len=	-0 TSval=2722
3	7 44.45575	000(192.168.6.41	192.168.7.41	IPA	63	5000	51711	unknown 0x2) [Malformed	d Packet]	
3	8 44.45579	000(192.168.7.41	192.168.6.41	TCP	52	51711	5000	51711 > 500	[ACK] Seq=	=4 Ack=12 Win=29312 Ler	=0 TSva1=274
3	9 48.81285	500(192.168.7.41	192.168.6.41	IPA	59	51711	5000	unknown 0x4	[Malformed	d Packet]	
4	0 48.85175	100(192.168.6.41	192.168.7.41	IPA	63	5000	51711	unknown 0x2) [Malformed	d Packet]	
4	1 48.85180	800(192.168.7.41	192.168.6.41	TCP	52	51711	5000	51711 > 500	[ACK] Seq=	=11 Ack=23 Win=29312 Le	n=0 TSval=2
4	2 53.75263	000(192.168.7.42	192.168.6.42	IPA	62	53732	\$ 5000	unknown 0x6	: [Malformed	d Packet]	
4	3 53.79189	400(192.168.6.42	192.168.7.42	TCP	52	5000	53732	5000 > 5373	[ACK] Seq	=1 Ack=11 Win=14592 Ler	=0 TSval=272
4	4 53.79197	800(192.168.6.42	192.168.7.42	IPA	63	5000	53732	unknown 0x2) [Malformed	d Packet]	
4	5 53.79204	100(192.168.7.42	192.168.6.42	TCP	52	53732	5000	53732 > 500	[ACK] Seq=	=11 Ack=12 Win=29312 Le	n=0 TSval=2
4	6 56.91860	500(192.168.7.42	192.168.6.42	IPA	56	53732	5000	unknown 0x6	[Malformed	d Packet]	
4	7 56.95971	700(192.168.6.42	192.168.7.42	IPA	68	5000	53732	unknown 0x6	[Malformed	d Packet]	
4	8 56.95979	400(192.168.7.42	192.168.6.42	TCP	52	53732	5000	53732 > 500	[ACK] Seq=	=15 Ack=28 Win=29312 Le	n=0 TSval=2
4	9 56.95998	600(192.168.7.42	192.168.6.42	TCP	52	53732	5000	53732 > 500	[FIN, ACK]] Seq=15 Ack=28 Win=293	12 Len=0 TSV
5	0 57.00030	900(192.168.6.42	192.168.7.42	IPA	63	5000	53732	unknown 0x2) [Malformed	d Packet]	
5	1 57.00036	200(192.168.7.42	192.168.6.42	TCP	40	53732	5000	53732 > 500	[RST] Seq=	=16 Win=0 Len=0	
5	2 57.00080	200(192.168.6.42	192.168.7.42	IPA	1500	5000	53732	unknown 0x2) [Malformed	d Packet]	
5	3 57.00083	700(192.168.7.42	192.168.6.42	TCP	40	53732	5000	53732 > 500	[RST] Seq=	=16 Win=0 Len=0	
5	4 57.00090	000(192.168.6.42	192.168.7.42	IPA	1500	5000	53732	unknown 0x6	<pre>[Malformed</pre>	d Packet1	
Frame	34: 40 by	tes on wire (320 bit	s), 40 bytes captured	(320 bits)							
Raw p	acket data										
Inter	net Protoc	ol Version 4, Src: 1	92.168.7.43 (192.168.7	.43), Dst: 19	2.168.6.	43 (192.168	3.6.43)				
Trans	mission Co	ntrol Protocol, Src	Port: 56691 (56691), [ost Port: 5000	(5000),	Seq: 15, L	.en: 0				
000 4	15 00 00 29	04 20 40 00 40 06	c7 of c0 a8 07 2b E	(00 0							
010 0	0 a8 06 26	dd 73 13 88 9f ac	24 6d 00 00 00 00	+ s \$m							
020 5	0 04 00 00	6c 24 00 00	P	1\$							

Figure 6: Wireshark capture for tun (client) at tun2 interface