CS 536 Fall 2016: Lab4

Maria L. Pacheco

November 1, 2016

1 Problem 1

Since `tunneld` had to work for a variety of scenarios: clients that send consecutive packages, like the traffic generation application, and clients that send a package and expect a direct response, like the ping application, we needed to provide an implementation that was agnostic to the protocol that the original server and client were using to control their communication. For this reason, I implemented timeouts using alarms for the `recvfrom` calls in `tunneld`. This way, if the server doesn't respond, we can continue to send the packages that come from the client and viceversa. The timeout used was of **1 second**, which will incur in additional overhead in the times that will be discussed below, apart from the overhead of the additional hop.

In table 1 we can see the differences between running the traffic generation application with tunneling and without tunneling.

	Receiver		
Mode	time	Mbps	pps
Tunneling	1.000012	0.01	1.00
Tunneling (- 1 sec)	0.000012	689.33	83333.33
No Tunneling	0.000006	1323.52	167772.16

Table 1: Traffic generation comparison: tunneling and no tunneling

We can observe a difference in performance even by subtracting the alarm overhead. This means the additional hops needed to go through the VPN have a clear impact in the time and performance of the communication between client and server.

In table 2 we can see the differences between running the ping application with tunneling and without tunneling. In this case we don't care about the 1 sec overhead as the server does respond to the client and the call to `recvfrom` doesn't wait for a second. We can again observe a clear impact of the VPN in the performance.

Mode	Time
Tunneling	0.922119
No Tunneling	0.439941

Table 2: Ping comparison: tunneling and no tunneling

In both cases, the VPN duplicated the time needed for communication.

2 Problem 3

In figure 1 we can observe some information of the first 3 frames: the Ethernet, IP and UPD headers as well as the first few bytes of the UPD payload. Ethernet header is marked in blue, IP header in red, UPD header in green and the rest corresponds to the UDP payload.

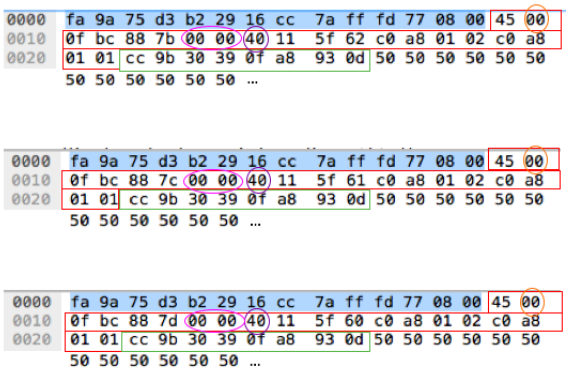

Figure 1: 3 first sniffed frames

We can see that the value of the TTL field is set to 0x40 which is 64 in decimal for all three packets. By doing ping to `www.purdue.edu` 3 and to `www.cisco.com` 2 on the same machine we can observe different values for the TTL field in all cases. This makes sense as the field is associated with: 1) the value set initially by the kernel at the sender and 2) the number of hops the packages do to reach its destination. The TTL field will be decreased by 1 every time it reaches a router that is not the destination.

Figure 2: Ping results to Cisco

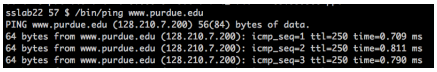

Figure 3: Ping results to Purdue

Different operating systems and different kernel versions have different initial values for the TTL field. This can be exploited by attackers as a tool to find out the OS and even its version of their target. Attackers also can send consecutive packages by continuously incrementing TTL values from 1 to find out the route that packages are following, as routers return "Time exceeded" messages when

the TTL reaches 0. This gives attackers information about the IP addresses of the hops packages are taking to reach a target.

TOS fields are replaced by "Differentiated Services Field", but in these packages it is set to 0x00, which is the default value. Fragmentation fields are also set to 0x0000, which means they are not being used.