TCP handshake with SOCK_RAW socket

Asked 16 years, 3 months ago Modified 12 years, 1 month ago

Viewed 16k times

20

Ok, I realize this situation is somewhat unusual, but I need to establish a TCP connection (the 3-way handshake) using only raw sockets (in C, in linux) -- i.e. I need to construct the IP headers and TCP headers myself. I'm writing a server (so I have to first respond to the incoming SYN packet), and for whatever reason I can't seem to get it right. Yes, I realize that a SOCK_STREAM will handle this for me, but for reasons I don't want to go into that isn't an option.

The tutorials I've found online on using raw sockets all describe how to build a SYN flooder, but this is somewhat easier than actually establishing a TCP connection, since you don't have to construct a response based on the original packet. I've gotten the SYN flooder examples working, and I can read the incoming SYN packet just fine from the raw socket, but I'm still having trouble creating a valid SYN/ACK response to an incoming SYN from the client.

So, does anyone know a good tutorial on using raw sockets that goes beyond creating a SYN flooder, or does anyone have some code that could do this (using

SOCK_RAW, and not SOCK_STREAM)? I would be very grateful.

MarkR is absolutely right -- the problem is that the kernel is sending reset packets in response to the initial packet because it thinks the port is closed. The kernel is beating me to the response and the connection dies. I was using tcpdump to monitor the connection already -- I should have been more observant and noticed that there were TWO replies one of which was a reset that was screwing things up, as well as the response my program created. D'OH!

The solution that seems to work best is to use an iptables rule, as suggested by MarkR, to block the outbound packets. However, there's an easier way to do it than using the mark option, as suggested. I just match whether the reset TCP flag is set. During the course of a normal connection this is unlikely to be needed, and it doesn't really matter to my application if I block all outbound reset packets from the port being used. This effectively blocks the kernel's unwanted response, but not my own packets. If the port my program is listening on is 9999 then the iptables rule looks like this:

```
iptables -t filter -I OUTPUT -p tcp --sport 9999 -
-tcp-flags RST RST -j DROP
```

linux **network-programming**

Share

Improve this question

Follow

edited Dec 5, 2011 at 11:41

Kev

120k ● 53 ● 305 ● 391

asked Sep 21, 2008 at 5:43

azure

201 ● 1 ● 2 ● 5

7 Answers

Sorted by: Highest score (default)

12

You want to implement part of a TCP stack in userspace... this is ok, some other apps do this.

One problem you will come across is that the kernel will be sending out (generally negative, unhelpful) replies to incoming packets. This is going to screw up any communication you attempt to initiate.

One way to avoid this is to use an IP address and interface that the kernel does not have its own IP stack using- which is fine but you will need to deal with link-layer stuff (specifically, arp) yourself. That would require a socket lower than IPPROTO_IP, SOCK_RAW - you need a packet socket (I think).

It may also be possible to block the kernel's responses using an iptables rule- but I rather suspect that the rules will apply to your own packets as well somehow, unless you can manage to get them treated differently (perhaps applying a netfilter "mark" to your own packets?)

Read the man pages

socket(7) ip(7) packet(7)

Which explain about various options and ioctls which apply to types of sockets.

Of course you'll need a tool like Wireshark to inspect what's going on. You will need several machines to test this, I recommend using vmware (or similar) to reduce the amount of hardware required.

Sorry I can't recommend a specific tutorial.

Good luck.

Share Improve this answer

Follow

answered Sep 21, 2008 at 6:57

MarkR

63.5k ● 15 ● 119 ● 154

4

I realise that this is an old thread, but here's a tutorial that goes beyond the normal SYN flooders: http://www.enderunix.org/docs/en/rawipspoof/

Hope it might be of help to someone.

Share Improve this answer

Follow

answered Jul 30, 2012 at 11:30

youjustreadthis

652 ● 3 ● 10 ● 25

I can't help you out on any tutorials.

2

But I can give you some advice on the tools that you could use to assist in debugging.

First off, as bmdhacks has suggested, get yourself a copy of wireshark (or tcpdump - but wireshark is easier to use). Capture a good handshake. Make sure that you save this.

Capture one of your handshakes that fails. Wireshark has quite good packet parsing and error checking, so if there's a straightforward error it will probably tell you.

Next, get yourself a copy of tcpreplay. This should also include a tool called "tcprewrite". tcprewrite will allow you to split your previously saved capture files into two - one for each side of the handshake. You can then use tcpreplay to play back one side of the handshake so you have a consistent set of packets to play with.

Then you use wireshark (again) to check your responses.

Share Improve this answer

Follow

edited May 23, 2017 at 11:48

Community Bot

1 • 1

answered Sep 21, 2008 at 6:44

Andrew Edgecombe

40.3k • 3 • 38 • 63

I don't have a tutorial, but I recently used Wireshark to good effect to debug some raw sockets programming I was doing. If you capture the packets you're sending,

1

wireshark will do a good job of showing you if they're malformed or not. It's useful for comparing to a normal connection too.

Share Improve this answer

Follow

answered Sep 21, 2008 at 5:51

bmdhacks

16.3k ● 8 ● 36 ● 55

0

There are structures for IP and TCP headers declared in netinet/ip.h & netinet/tcp.h respectively. You may want to look at the other headers in this directory for extra macros & stuff that may be of use.

You send a packet with the SYN flag set and a random sequence number (x). You should receive a SYN+ACK from the other side. This packet will have an acknowledgement number (y) that indicates the next sequence number the other side is expecting to receive as well as another sequence number (z). You send back an ACK packet that has sequence number x+1 and ack number z+1 to complete the connection.

You also need to make sure you calculate appropriate TCP/IP checksums & fill out the remainder of the header for the packets you send. Also, don't forget about things like host & network byte order.

TCP is defined in RFC 793, available here: http://www.faqs.org/rfcs/rfc793.html

Share Improve this answer

edited Sep 21, 2008 at 7:39

Follow

answered Sep 21, 2008 at 7:22

JB

0

Depending on what you're trying to do it may be easier to get existing software to handle the TCP handshaking for you.

One open source IP stack is lwIP (http://savannah.nongnu.org/projects/lwip/) which provides a full tcp/ip stack. It is very possible to get it running in user mode using either SOCK_RAW or pcap.

Share Improve this answer

Follow

answered Jun 6, 2009 at 14:53

Tom Hennen

4,904 ● 7 ● 36 ● 46

0

if you are using raw sockets, if you send using different source mac address to the actual one, linux will ignore the response packet and not send an rst.

Share Improve this answer

Follow

answered Nov 2, 2012 at 16:07

A G

1,117 ● 2 ● 22 ● 36

I know it's 4 years old question but your suggestion looks quite interesting so I decided to ask you for clarification. The

question is: Is there any problem with the router while routing response packet to certain machine with wrong device's MAC address? Will the packet be filtered by nowaday firewall or antivirus? – David Tran Jan 10, 2016 at 10:44

A spoofed MAC might be filtered by switches if they are configured for port security. There might be ACLs on the MAC layer, but normally there is nothing stopping you from spoofing MACs. You only need to make sure your network card will accept answers to this spoofed mac (by adding the mac to the table of accepted packets or by putting it in promisc mode). This is a bad idea to do this if you are not authorized. – eckes Nov 10, 2017 at 6:05