IPgrab

Verbose Packet Sniffer
Edition 0.9.3, for IPgrab version 0.9.3
18 May 2001

Mike Borella

Table of Contents

1 Introduction

You don't really understand networks until you've watched traffic on the wire. I believe strongly in this statement. In fact, I believe in it so strongly that I've been devleoping a packet sniffer so that all of us can learn about networking this way.

A packet sniffer is an application layer program that interacts with one or more layer two or layer three kernel modules or device drivers to capture packets on a network. The lower-layer pieces read the packet off the wire, copy it into memory, and provide an API for an application to read it. An application, such as IPgrab, can do whatever it likes with the resulting image of a packet. Packet sniffers have been used for many years to detect network problems, troubleshoot protocols, and detect intruders.

Traditionally, packet sniffers have displayed the captured packets in brief, rather cryptic formats.

IPgrab is my humble attempt to make most, and eventually all, network protocols readable. It currently supports a wide variety of IP-related protocols, including some of the newer IP telephony protocols such as SIP and MGCP, as well as IPv6. IPgrab also decodes basic IPX and NETBIOS packets.

The center of all IPgrab development, testing, and communication is hosted on SourceForge at `http://ipgrab.sourceforge.net/`. You may mail suggestions and bug reports for to me at `mike@borella.net`.

2 Guidelines for Use

IPgrab can be used in a number of ways, for a number of purposes. In this section we provide a brief overview of IPgrab's two modes and its command-line options.

2.1 Main Mode

Main mode is the default mode for IPgrab output. It is extremely verbose, displaying each field from all packet headers and protocols that it understands across a separate line of text. Banners separate different layers of protocol output. Single packets may require more than 100 lines in order to be displayed. Main mode is most useful if you need to know why or when a certain field or fields take on certain values. Below is an example of main mode formatting of a TCP packet.

```
**************************************************************************
                        Ethernet (990036574.132701)
--------------------------------------------------------------------------
Hardware source:        00:80:3e:57:b4:cf
Hardware destination:   01:00:5e:00:01:16
Type / Length:          0x800 (IP)
Media length:           192
--------------------------------------------------------------------------
                        IP Header
--------------------------------------------------------------------------
Version:                4
Header length:          5 (20 bytes)
TOS:                    0x00
Total length:           178
Identification:         16
Fragmentation offset:   0
Unused bit:             0
Don't fragment bit:     0
More fragments bit:     0
Time to live:           29
Protocol:               17 (UDP)
Header checksum:        33315
Source address:         149.112.164.129
Destination address:    224.0.1.22
--------------------------------------------------------------------------
                        UDP Header
--------------------------------------------------------------------------
Source port:            1026 (unknown)
Destination port:       427 (SLP)
Length:                 158
Checksum:               17593
--------------------------------------------------------------------------
                        SLPv1 Header
--------------------------------------------------------------------------
Version:                1
```

```
Operation:              1 (service request)
Length:                 150
Flags/Reserved:         0x00
Dialect:                0
Language code:          en
Character encoding:     1000
XID:                    6365
```

In general, IPgrab does not attempt to "interpret" the values of a packet. For example, the IP TOS field is displayed in its raw value of 0x00. If there is an interpretation or further explanation of a field, IPgrab puts it in parenthesis following the raw value. For example, the IP header lenght field is displayed in its raw form of 5 followed by an interpretation of the number of bytes that this value represents. Likewise, the TCP source port is 23, which IPgrab recognizes as the telnet port.

Note that IPgrab adds a timestamp to the banner for each link layer packet.

2.2 Minimal Mode

IPgrab also supports a minimal mode in which all information about all parts of a packet are displayed in a single line of text. This line may be longer than 80 characters and thus wrap around a standard terminal window one or more times. Below is an example of minimal mode formatting of a TCP packet.

```
1 990038240.206509 | ETH 00:b0:d0:11:a4:d0->ff:ff:ff:ff:ff:ff | IP
149.112.90.171->149.112.90.255 (len:78,id:29629,frag:0) | UDP 137->137
| NETBIOS NS query 3-COM
```

Minimal mode begins with a number (in this case, the number 1 indicates that this packet is the first one read) and a timestamp, and then parses the packet from link layer to application layer. Each layer begins with an abbreviation of the protocol being displayed (such as ETH, IP, and UDP, above). These abbreviations are followed by only the most relevant fields of the protocol. For example, IP source and destination addresses are shown, along with the total length field and the DF bit (if set) in parentheses. Likewise, UDP source and destination ports are shown.

2.3 Command Line Options

Both main mode and minimal mode output can be adjusted by specifying one or more commend line options. In this section, we provide a complete list of IPgrab's command line options and their use.

The usage of IPgrab is briefly described as follows.

`ipgrab [-blmnPprTtwx] [-c|--count n] [-h|--help] [-i|--interface if] [BPF expr]`

The BPF expression is a string of terms that is acceptable to the Berkeley Packet Filter. For more details on the BPF expression grammar, see the tcpdump manual page.

- `-b`. Turn off buffering of standard output (stdout) so that all displaying occurs as soon as possible. Useful when IPgrab output is being re-directed to a file.
- `-c n / --count n`. Terminate after reading and displaying the first n packets.
- `-h / --help`. Display usage screen with a brief description of the command line options.

- `-i if / --interface if`. Makes IPgrab listen to packets on interface if. If this option is not used, the default interface will be assumed.
- `-l`. Don't display link layer headers.
- `-m`. Minimal mode output.
- `-n`. Don't display network layer headers.
- `-P`. Initiate a dynamic port mapping. This option must be followed by a string of the form '<protocol>=<port>', such as 'rtp=6569'.
- `-p`. Dump packet payloads beyond what IPgrab parse. In other words, if IPgrab doesn't parse a particular application, this option will dump the application data in hex and text format.
- `-r`. Read packets from a file, rather than an interface. The file should be created in "raw" format, such as with '`-w`' option.
- `-T`. Don't display timestamps in minimal mode.
- `-t`. Don't display transport layer headers.
- `-w`. Write the raw packets to a file, rather than the screen. The packets will not be parsed. The file can be read with the '`-r`' option.
- `-x`. Hex dump mode. After processing each layer, dump out the contents of that layer in hex and text. Only valid in main mode.

2.4 Examples

- Only ICMP packets will be displayed using main mode without link layer headers.

  Command: `ipgrab -l icmp`

  Output:

```
****************************************************************************
                        IP Header
----------------------------------------------------------------------------
Version:                4
Header length:          5 (20 bytes)
TOS:                    0x00
Total length:           84
Identification:         0
Fragmentation offset:   0
Unused bit:             0
Don't fragment bit:     1
More fragments bit:     0
Time to live:           64
Protocol:               1 (ICMP)
Header checksum:        42625
Source address:         149.112.90.225
Destination address:    198.147.221.66
----------------------------------------------------------------------------
                        ICMP Header
----------------------------------------------------------------------------
Type:                   8 (echo request)
```

```
Code:                   0
Checksum:               43361
Identifier:             15353
Sequence number:        0
```

- Only packets arriving on interface eth1 with a source port of 21 (FTP) will be displayed in minimal mode.

  Command: `ipgrab -i eth1 -m src port 21`

  Output:

  ```
  1 990038936.642292 | ETH 00:80:3e:57:b4:cf->00:50:04:32:0e:8f | IP
  198.147.221.66->149.112.90.225 (len:64,id:18353,DF,frag:0) | TCP 21->1047
  (SA,3123051349,2290714856,9856) <timestamp 873976824 43310056><window
  scale 0><SACK permitted><maximum segment size 1420>
  ```

3 Status of Protocol Modules

In this section we discuss the status of each of the protocol modules. While some protocols may be fully supported and well tested, other modules may only have partial support or may not have been tested fully.

3.1 Authentication Header (AH)

- Module: `ah.c`
- Support: Full.
- Maturity: Not tested.
- Notes: AH typically appears between IP and TCP/UDP headers in order to apply IPsec-based authentication.

3.2 Address Resolution Protocol (ARP)

- Module: `arp.c`
- Support: Partial (only supports IP over Ethernet).
- Maturity: Well tested.
- Notes: Originally, ARP was defined to support address resolution of network layer protocol x over link layer protocol y. Currently, IP over Ethernet is by far the most used mode of ARP, and thus is the only one supported.

3.3 Challenge Handshake Authentication Protocol (CHAP)

- Module: `chap.c`
- Support: Full.
- Maturity: Well tested.
- Notes: CHAP authenticates the ends of a PPP session to one another.

3.4 Dynamic Host Configuration Protocol (DHCP)

- Module: `dhcp.c`
- Support: Partial (doesn't support all options).
- Maturity: Well tested.
- Notes: DHCP is an extensibly protocol with a large number of officially sanctioned options, and a number of de facto options. Currently we support many of the most common, but not all, of these options.

3.5 Domain Name System (DNS)

- Module: `dns.c`
- Support: Partial (doesn't support all record types).
- Maturity: Well tested.
- Notes: Currently, we support records of type A, AAAA, CNAME, NS, SOA, and PTR. Other record types, such as A6, MX, and SRV, are not supported.

3.6 Encapsulating Security Payload (ESP)

- Module: `esp.c`
- Support: Partial (doesn't decrypt packets nor decode ESP trailer).
- Maturity: Not tested.
- Notes: ESP typically appears between IP and TCP/UDP headers in order to apply IPsec-based encryption and/or authentication. We do not attempt to decode encrypted packets because this would require adding state to IPgrab, which is something that we'd rather not do. This prevents us from displaying the ESP trailer as well. As soon as an ESP header is read and the plaintext portion is displayed, we halt processing of the rest of the packet.

3.7 Ethernet

- Module: `ethernet.c`
- Support: Full, expcept that not all Ethernet types are recognized, and 802.1p and VLANs are not supported.
- Maturity: Well tested.
- Notes: Supported Ethernet types include IP, IPv6, PPP, ARP, RARP, and IPX. LLC encapsulation is supported in the LLC module.

3.8 Generic Routing Encapsulation

- Module: `gre.c`
- Support: Full for versions 0 and 1.
- Maturity: Well tested.
- Notes: Version 1 is defined in the PPTP RFC.

3.9 Hypertext Transfer Protocol

- Module: `http.c`
- Support: Full (displays only headers).
- Maturity: Well tested.
- Notes: Parses and displays HTTP headers only. The rest of the payload is skipped.

3.10 Internet Control Message Protocol

- Module: `icmp.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Displays all ICMP types and codes, but does not parse specific payloads for some lesser-used ICMP types, such as source quench, and redirect. Also does not handle Mobile IP extensions (yet).

3.11 Internet Control Message Protocol Version 6

- Module: `icmpv6.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Not all types and codes are explicitly parsed.

3.12 Internet Group Message Protocol

- Module: `igmp.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: IGMPv1 and v2 are supported. IGMPv3 is not tested and may not be cleanly supported.

3.13 Internet Protocol

- Module: `ip.c`
- Support: Full.
- Maturity: Well tested.
- Notes: Full support for the IP header and options. TOS/DS byte is not interpreted in any particular fashion.

3.14 Internet Protocol Control Protocol

- Module: `ipcp.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Full support for common messages and options. Some options may not be supported.

3.15 Internet Protocol Version 6

- Module: `ipv6.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Full support for the IPv6 header, except that IPv6 addresses are not displayed in the proper shorthand.

3.16 Internet Packet Exchange

- Module: `ipx.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Full support for the IPX header, but not all transport protocols and applications are supported, nor are the header fields interpretated as well as they could be.

3.17 Internet Packet Exchange Routing Information Protocol

- Module: `ipxrip.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Basic listing of the routes.

3.18 Internet Key Exchange

See Internet Security Association and Key Management Protocol.

3.19 Internet Security Association and Key Management Protocol

- Module: `isakmp.c`
- Support: Partial.
- Maturity: Well tested against Windows 2000 only.
- Notes: Only the following ISAKMP headers are supported: delete, SA, vendor ID, proposal, transform.

3.20 Layer 2 Tunneling Protocol

- Module: `l2tp.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Only the most common message types are supported.

3.21 Layer Control Protocol

- Module: `lcp.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Not all NCPs are tested for.

3.22 Logical Link Control

- Module: `llc.c`
- Support: Partial.
- Maturity: Partially tested.
- Notes: Some basic cases such as IP and IPX encapsulation work reasonably well, but other common cases are not supported. Basically, this module needs a re-write.

3.23 Loopback

- Module: `loopback.c`
- Support: Full.
- Maturity: Fully tested.
- Notes: Some loopback interfaces, Redhat Linux 6.2 for example, present themselves as Ethernet interfaces, where all Ethernet addresses are zeroed out. Strange but true.

3.24 Media Gateway Control Protocol

- Module: `mgcp.c`
- Support: Partial.
- Maturity: Not tested.
- Notes: Use at your own risk.

3.25 Mobile IP

- Module: `mobileip.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Not all extensions are supported. Support for the CDMA2000 A11 interface is also in this module. We support registration update and registration acknowledgement, as well as some of the extensions.

3.26 NETBIOS Name Service

- Module: `netbios_ns.c`
- Support: Full.
- Maturity: Well tested.
- Notes: DNS-like protocol for NETBIOS.

3.27 Open Shortest Path First

- Module: `ospf.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Only hello messages are supported.

3.28 Point to Point Protocol

- Module: `ppp.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Most systems will not let you sniff native PPP packets, as their headers are usually stripped off before the kernel gives the packet to the sniffer. However, when you use PPTP or L2TP, PPP is avilable to a sniffer. Thus all testing was done using tunneling modes, rather than native mode. Note that some tunnel configurations may fragment a single incoming PPP frame into multiple tunneled packets. In this case, it is not clear what ipgrab will do (probably something strange). We currently do not decode HDLC-mode (control escape) PPP packets.

3.29 PPP Over Ethernet

- Module: `pppoe.c`
- Support: Full.
- Maturity: Not tested.
- Notes: This is a contributed module. I have not tested it, but the contributor has.

3.30 Point to Point Tunneling Protocol

- Module: `pptp.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Not all message types are supported, but the most common ones are.

3.31 Raw IP

- Module: `raw.c`
- Support: Full.
- Maturity: Well tested.
- Notes: This is a default datalink type for native IP. Since most kernel don't let us look at native PPP frames, most packets on a PPP interface will be interpreted as raw.

3.32 Routing Information Protocol

- Module: `rip.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: RIPv1 was tested extensively. RIPv2 was not tested.

3.33 Routing Information Protocol (Next Generation)

- Module: `ripng.c`
- Support: Full.
- Maturity: Well tested.
- Notes:

3.34 Real Time Protocol

- Module: `rtp.c`
- Support: Partial.
- Maturity: Not tested.
- Notes: n order to use this module, you'll need to notify IPgrab of proper port number on which to expect the RTP traffic. Use the `-P` option to do this.

3.35 Session Description Protocol

- Module: `sdp.c`
- Support: Full.
- Maturity: Well tested.
- Notes: Displays the headers in plain format with no interpretation. Does not display anything in minimal mode.

3.36 Session Initiation Protocol

- Module: `sip.c`
- Support: Full.
- Maturity: Well tested.
- Notes: Displays the headers in plain format with no interpretation.

3.37 Serial Line IP

- Module: `slip.c`
- Support: Full.
- Maturity: Not tested.
- Notes:

3.38 Service Location Protocol

- Module: `slp.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Only basic forms of version 1 are supported.

3.39 Simple Network Management Protocol

- Module: `snmp.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: This module only displays the most basic information about captured SNMP packets.

3.40 Sequenced Packet Exchange

- Module: `spx.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Not all applications are supported.

3.41 Secure Shell

- Module: `ssh.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Only the initial version number exchange is supported.

3.42 Transmission Control Protocol

- Module: `tcp.c`
- Support: Partial.
- Maturity: Well tested.
- Notes: Not all options are well supported. Program might crash on assorted nastygrams. In the minimal mode output, there are four parameters following the port numbers. They are, in order: a list of all flags that are set, the sequence number, the acknowledgement number, and the advertized window size.

3.43 User Datagram Protocol

- Module: `udp.c`
- Support: Full.
- Maturity: Fully tested.
- Notes:

4 APIs

The internal structure of IPgrab is logically divided into modules, one per protocol. For example, IP headers and options are decoded and displayed in `ip.c`, TCP headers and options are decoded and displayed in `tcp.c`, and so on.

IPgrab modules use two APIs – the first for reading data from a packet that has been captured, and the second for displaying this data (or some derivation thereof) to an output device.

In this chapter, we'll explore and document these APIs.

4.1 Reading APIs

4.2 Writing APIs

IPgrab supports a rich set of APIs for displaying data of various types in a number of formats. There are two basic types of output for IPgrab: main mode, or minimal mode. In main mode, each piece of data (e.g., a field in a protocol header) is formatted

– `display(char * label, u_int8_t * content, u_int8_t length, display_t format)`

5 History

Like many other folks, I started using `tcpdump` after reading Rich Stevens' wonderful book, *TCP/IP Illustrated Vol. 1.* Each packet is summarized in a single compact, but slightly cryptic, line of output. While `tcpdump` remained a classic, around 1997 development had slowed quite a bit. Support for new protocols was not being added to the official distribution, and understanding and modifying the existing code could be trying. I felt the need to provide a more general packet sniffer that not only displayed *all* of a packet's fields, but was written in a way that could easily be read, understood, and modified.

In Fall 1997 I developed the first few versions of IPgrab. They were very tentative, just displaying Ethernet, IP, TCP, and UDP fields. For the most part, they only compiled on Linux. But I had gotten a taste of how useful such a tool could be. I used IPgrab to find out that Windows NT 4.0 incremented IP identification fields by 256 instead of 1, and to find out that a LAN router wasn't proxy ARPing correctly.

Over the next two years I added features to IPgrab. Most of the work was done in my spare time, and progress was slow. Occasionally I received a very useful patch from a user. I also worked on porting it to other systems besides Linux – in particular, FreeBSD and Solaris. By mid-1999, IPgrab was stable (version 0.8.2) and supported a number of rather complex protocol suites, such as IPsec, L2TP and some VoIP protocols. I didn't do much work until April 2000, when I decided to host it on Sourceforge.net.

It was time for a massive overhaul. Development took three major angles: (1) New APIs for safe reading from a packet and displaying to an output device, (2) a line-based minimal output mode comparable to `tcpdump`, and (3) more protocol support. In particular, the APIs took a while to get right, but now they're in place and work really well. This required a re-write of every module, resulted in an overall cleanup of the code. Release 0.9 was the first official release with these new features.

`tcpdump` development is once again underway and there are many freeware and open source packet sniffers available that do much of what IPgrab does. However, I've continued to develop IPgrab for a number of reasons. In particular, new protocols are being designed so quickly by the IETF and other organizations that it becomes very useful to be able to add these protocols quickly to a sniffer. Some of the extensions to IETF protocols that are defined by other Standards Development Organizations are typically not supported in sniffers. Also, developing IPgrab gives me a reason to stay on top of protocol developments and keep my hands dirty with coding.

6 Index

A

C

D

E

G

H

I

L

M

N

O

P