PcapNG File Format

by Erik Hjelmvik

The PcapNG file format (aka "PCAP Next Generation", "pcap-ng" or ".pcapng") is a capture file format designed to overcome limitations in the original libpcap file format, such as the inability to store packets with different link layer types.

> *If you've come here to convert a PcapNG file to PCAP, then check out the blog post How to handle PcapNG files instead.*

The original libpcap file format, which is often referred to as just "PCAP", was created by Van Jacobson, Craig Leres and Steven McCanne around 1987 as part of the work they did on tcpdump and libpcap. The PCAP file format supported storing packet records, which contained a timestamp, length and the data for each captured packet.

The PcapNG file format was born around 2004, partly as a result of an email thread regarding a "proposed new pcap format" on the tcpdump mailing list. In 2005 Gianluca Varenni and a couple of additional WinPcap developers published the NTAR library, which implemented the "PCAP Next Generation Dump File Format" that had been discussed on the tcpdump mailing list.

Some of the prominent features introduced with the PcapNG format are:

- Ability to store packets with different link layer types, such as Ethernet and 802.11 WiFi packets, in the same capture file.
- Multiple PcapNG capture files can be merged into one simply by concatenating them.
- Allows comments, also known as "annotations", to be attached to packets.
- Data structures designed specifically for storing metadata, such as hostnames, encryption keys, capture filters and information about the interface used to capture the packets.

The PcapNG file format became the default save-file format for Wireshark in 2012, with the release of version 1.8. In practice this also meant that capture files generated with command line tools like tshark, mergecap, editcap and dumpcap also used the PcapNG file format unless the "-F pcap" or "-P" argument was used. Wireshark's switch from PCAP to PcapNG is the primary driver behind the proliferation of .pcapng files we're seeing today.

In the sections below I cover the core features of the PcapNG structure, which are the bare minimum required to handle network traffic in .pcapng files. For the full PcapNG specification, please see the IETF RFC draft specification.

Section Header Block

A PcapNG file must start with a Section Header Block (SHB), which has the following structure:

- Block Type (4 bytes) = 0x0a0d0d0a
- Block Length (4 bytes)
- Byte-Order Magic (4 bytes) = 0x1a2b3c4d
- Major Version (2 bytes) = 0x0001
- Minor Version (2 bytes) = 0x0000
- Section Length (8 bytes) = 0xffffffffffffffff
- Options (variable length)
- Block Length (redundant 4 byte value)

Here's an example of an actual SHB:

```
0a 0d 0d 0a 8c 00 00 00 4d 3c 2b 1a 01 00 00 00  ........M<+.....
ff ff ff ff ff ff ff ff 03 00 2d 00 4d 61 63 20  ..........-.Mac 
4f 53 20 58 20 31 30 2e 31 30 2e 34 2c 20 62 75  OS X 10.10.4, bu
69 6c 64 20 31 34 45 34 36 20 28 44 61 72 77 69  ild 14E46 (Darwi
6e 20 31 34 2e 34 2e 30 29 00 00 00 04 00 34 00  n 14.4.0).....4.
44 75 6d 70 63 61 70 20 31 2e 31 32 2e 36 20 28  Dumpcap 1.12.6 (
76 31 2e 31 32 2e 36 2d 30 2d 67 65 65 31 66 63  v1.12.6-0-gee1fc
65 36 20 66 72 6f 6d 20 6d 61 73 74 65 72 2d 31  e6 from master-1
2e 31 32 29 00 00 00 00 8c 00 00 00              .12)........
```

- 0a 0d 0d 0a = Block Type is SHB
- 8c 00 00 00 = Block Length is 140 bytes
- 4d 3c 2b 1a = Section Byte Order is little endian
- 01 00 = Major Version is 1
- 00 00 = Minor Version is 0
- ff ff ff ... = Section Length is unknown
- 03 00 2d ... = Options
- 8c 00 00 00 = Block Length is 140 bytes

The **block type** field in SHB is also the PcapNG file type magic number, since it always appears first in a PcapNG file. The byte sequence that defines PcapNG files is "0a 0d 0d 0a". This particular sequence was chosen for two reasons:

- The byte sequence is a palindrome, i.e. it has the same value (0x0a0d0d0a) regardless if it's read as a little-endian or big-endian value.

- The sequence is built from line feed characters (newline and carriage return) so that an ASCII mode FTP transfer (RFC 959) of a PcapNG file will corrupt the PcapNG file header, unless the FTP application properly handles newline characters in ASCII transfer mode.

As you might have guessed, the **block length** field is a 32 bit number indicating the size of the SHB. This number can be stored either using big endian or little endian encoding. It is therefore crucial to read the next field (byte-order magic) before parsing the length field.

The **byte-order magic** field always has the value 0x1a2b3c4d and is encoded in the endianness used throughout the current section, which lasts until a new SHB is encountered or the PcapNG file ends. This means that the byte-order magic is encoded as "4d 3c 2b 1a" in a little endian section and "1a 2b 3c 4d" in a big endian section.

The **major and minor version** numbers have been 0x0001 and 0x0000 (i.e. version 1.0) ever since the first NTAR draft specification was published in 2004, but other values could in theory be used in the future.

The **section length** is a signed 64 bit number that is supposed to specify the length of the current section, i.e. the number of bytes until the next SHB or end of file. However, applications that output PcapNG files typically don't know beforehand how large the current section will be. The section length is therefore normally -1 (0xffffffffffffffff), which means that the size of the section isn't specified.

The **options** part of the SHB typically includes information about the hardware, operating system and application that was used to create the PcapNG file. Below is a typical example of SHB options.

```
02 00 37 00 49 6e 74 65 6c 28 52 29 20 43 6f 72  ..7.Intel(R) Cor
65 28 54 4d 29 20 69 35 2d 36 34 34 30 48 51 20  e(TM) i5-6440HQ 
43 50 55 20 40 20 32 2e 36 30 47 48 7a 20 28 77  CPU @ 2.60GHz (w
69 74 68 20 53 53 45 34 2e 32 29 00 03 00 1e 00  ith SSE4.2).....
36 34 2d 62 69 74 20 57 69 6e 64 6f 77 73 20 31  64-bit Windows 1
30 2c 20 62 75 69 6c 64 20 31 34 33 39 33 00 00  0, build 14393..
04 00 2e 00 44 75 6d 70 63 61 70 20 28 57 69 72  ....Dumpcap (Wir
65 73 68 61 72 6b 29 20 32 2e 36 2e 31 20 28 76  eshark) 2.6.1 (v
32 2e 36 2e 31 2d 30 2d 67 38 36 30 61 37 38 62  2.6.1-0-g860a78b
33 29 00 00 00 00 00 00                          3)......
```

- 02 00 = shb_hardware
- 37 00 = Hardware name length is 55 bytes
- 49 6e ... = "Intel(R) Core(TM) i5-6440HQ CPU @ 2.60GHz (with SSE4.2)"
- 00 = padding
- 03 00 = shb_os
- 1e 00 = OS name length is 30 bytes
- 36 34 ... = "64-bit Windows 10, build 14393"
- 00 00 = padding
- 04 00 = shb_userappl
- 00 2e = User application name length is 46 bytes
- 44 75 ... = "Dumpcap (Wireshark) 2.6.1 (v2.6.1-0-g860a78b3)"

- 00 00 = padding
- 00 00 = opt_endofopt
- 00 00 = padding

The **trailing block length** value is a duplicate of the one in the beginning of the block, and should hold the same value.

Interface Description Block

The first block to follow after the SHB is often an Interface Description Block (IDB), which contains metadata about the network interface used to capture the packets. A PcapNG file may contain multiple IDBs, one for each interface used to capture traffic. These IDBs don’t necessarily need to be placed at the beginning of the PcapNG file though, new IDBs can appear anywhere in a PcapNG file.

- Block Type (4 bytes) = 0x00000001
- Block Total Length (4 bytes)
- Link Type (2 bytes)
- Reserved (2 bytes) = 0x0000
- Snap Length (4 bytes)
- Options (variable length)
- Block Total Length (redundant 4 byte value)

Here's an example of an actual SHB:

```
01 00 00 00 d4 00 00 00 01 00 00 00 00 00 04 00  ................
02 00 32 00 5c 44 65 76 69 63 65 5c 4e 50 46 5f  ..2.\Device\NPF_
7b 44 46 41 33 36 34 45 35 2d 34 41 39 34 2d 34  {DFA364E5-4A94-4
42 35 38 2d 42 44 39 44 2d 36 31 37 41 32 43 39  B58-BD9D-617A2C9
38 35 39 38 39 7d 00 00 03 00 29 00 45 78 74 65  85989}....).Exte
72 6e 61 6c 20 36 33 2e 32 33 37 2e 32 33 33 2e  rnal 63.237.233.
36 30 20 28 61 6b 61 20 31 39 32 2e 31 36 38 2e  60 (aka 192.168.
35 2e 36 30 29 00 00 00 09 00 01 00 06 00 00 00  5.60)..........
0b 00 15 00 00 69 70 20 68 6f 73 74 20 39 36 2e  .....ip host 96.
39 33 2e 31 30 37 2e 33 34 00 00 00 0c 00 29 00  93.107.34.....).
36 34 2d 62 69 74 20 57 69 6e 64 6f 77 73 20 53  64-bit Windows S
65 72 76 65 72 20 32 30 31 32 20 52 32 2c 20 62  erver 2012 R2, b
75 69 6c 64 20 39 36 30 30 00 00 00 00 00 00 00  uild 9600.......
d4 00 00 00                                      ....
```

- 01 00 00 00 = Block Type is IDB
- d4 00 00 00 = Block Length is 212 bytes
- 01 00 = Link Type is Ethernet
- 00 00 = Reserved
- 00 00 04 00 = Snap Length is 256 kB
- Options
  - 02 00 = if_name
  - 32 00 = Interface name length is 50 bytes
  - 5c 44 65 ... = "\Device\NPF_{DFA364E5-4A94-4B58-BD9D-617A2C985989}"
  - 00 00 = padding
  - 03 00 = if_description

  - 29 00 = Interface description length is 41 bytes
  - 45 78 74 ... = "External 63.237.233.60 (aka 192.168.5.60)"
  - 00 00 00 = padding
  - 09 00 = if_tsresol
  - 01 00 = Time resolution length data is 1 byte
  - 06 = Time resolution is Microseconds (10^-6)
  - 00 00 00 = padding
  - 0b 00 = if_filter
  - 15 00 = Filter length is 21 bytes
  - 00 = Filter is a BPF string
  - 69 70 20 = "ip host 96.93.107.34"
  - 00 00 00 = padding
  - 0c 00 = if_os
  - 29 00 = Interface OS length is 41 bytes
  - 36 34 2d = "64-bit Windows Server 2012 R2, build 9600"
  - 00 00 00 = padding
  - 00 00 = opt_endofopt
  - 00 00 = padding
- d4 00 00 00 = Block length is 212 bytes

The **link type** value defines which type of packets the network interface captures, for example Ethernet or WiFi. A list of all link layer type values is available on the tcpdump website.

The **snap length** value is a 32 bit number indicating the maximum packet size that can be stored without truncating the packet data. This value is often 0x0000ffff (65535 bytes), 0x00040000 (256 kB) or 0x00000000 (no limit), but can in theory be any value.

The **options** section typically contains information about the network interface, such as:

- The interface name, like "eth0" or "\Device\NPF_{D007FF28-3F24-4C1B-8EF5-069A6FB811ED}"
- A descriptive interface name, like "Ethernet 2" or "VMware Network Adapter VMnet8"
- The IP address of the interface
- The MAC address of the interface
- The filter used when capturing, like "port 443 and host 1.2.3.4"
- The operating system of the sniffer, like "Mac OS X 10.15.7" or "Linux 4.15.0-76-generic"

There is also an important option called "**if_tsresol**" (option code 0x0009), which specifies the time unit used in packet timestamps. If no if_tsresol option is provided then the default microsecond time unit is used. Many capture files use the if_tsresol option to specify nanosecond resolution, which is encoded as if_tsresol = 0x09. It is also possible to specify timestamp resolution as a binary fraction of a second, but that encoding of if_tsresol is rarely (or never?) used so that's out of scope for this article. It's unfortunate that the time resolution is defined in IDB, since timestamps are also used in block types that don't refer to any specific interface. It is therefore not always clear which time resolution that should be used when parsing timestamps in non-packet block types like

GPS Blocks or Process Event Blocks. It would have made more sense to define the time resolution in SHB instead, or to specify it separately for each timestamp value.

Packet Blocks

The IETF draft specification for the PcapNG file format defines three block types that can be used to store packets, which are "Packet Block", "Simple Packet Block" (SPB) and "Enhanced Packet Block" (EPB). The first one (Packet Block) is, however, marked "obsolete" in the specification. The second packet block type (SPB) lacks support for metadata, such as timestamps, which makes it quite useless. This means that in practice only EPB is used to store packets in PcapNG files.

The format for EPB is as follows:

- Block Type (4 bytes) = 0x00000006
- Block Total Length (4 bytes)
- Interface ID (4 bytes)
- Timestamp Upper (4 bytes)
- Timestamp Lower (4 bytes)
- Captured Packet Length (4 bytes)
- Original Packet Length (4 bytes)
- Packet Data (variable length)
- Options (variable length)
- Block Total Length (redundant 4 byte value)

The **Interface ID** value refers to a particular Interface Description Block by referencing its implicit zero-indexed IDB position in the current section. I.e. the first IDB defined in a section has Interface ID 0x00000000, the second one has 0x00000001, and so on. This feature allows each individual packet to be tagged to the interface it was received on. This is also the feature that allows PcapNG files to support packets with mixed link layer types in the same capture file, which is something the original PCAP file format didn't support.

The two 32 bit **timestamp** fields are actually part of a single 64 bit timestamp. But contrary to other 64 bit fields, such as section length in SHB, this value has been split in half with the upper 32 bits stored first, followed by the lower 32 bits. This means that the EPB timestamps that are stored using big endian encoding can actually be read as a coherent 64 bit value, while little endian ones have to be read as two separate 32 bit fields. The combined 64 bit value from these two fields is the number of milliseconds that have elapsed since 1 January 1970, unless a custom timestamp resolution has been defined in the tagged IDB using the if_tsresol option.

The **captured** and **original packet length** fields are kept unchanged compared to the original PCAP file format, where the captured length field indicates the number of bytes of packet data and the original length field indicates the size of the actual packet on the network. These two values are typically the same unless a short snap length is used.

The **packet data** section contains the actual packet data that was transferred over the network. This data is stored exactly as it was received, without caring about endianness or correctness of the data.

The options section in EPB is sometimes used to store per-packet comments, also known as “annotations”, using the option code 0x0001 (opt_comment).

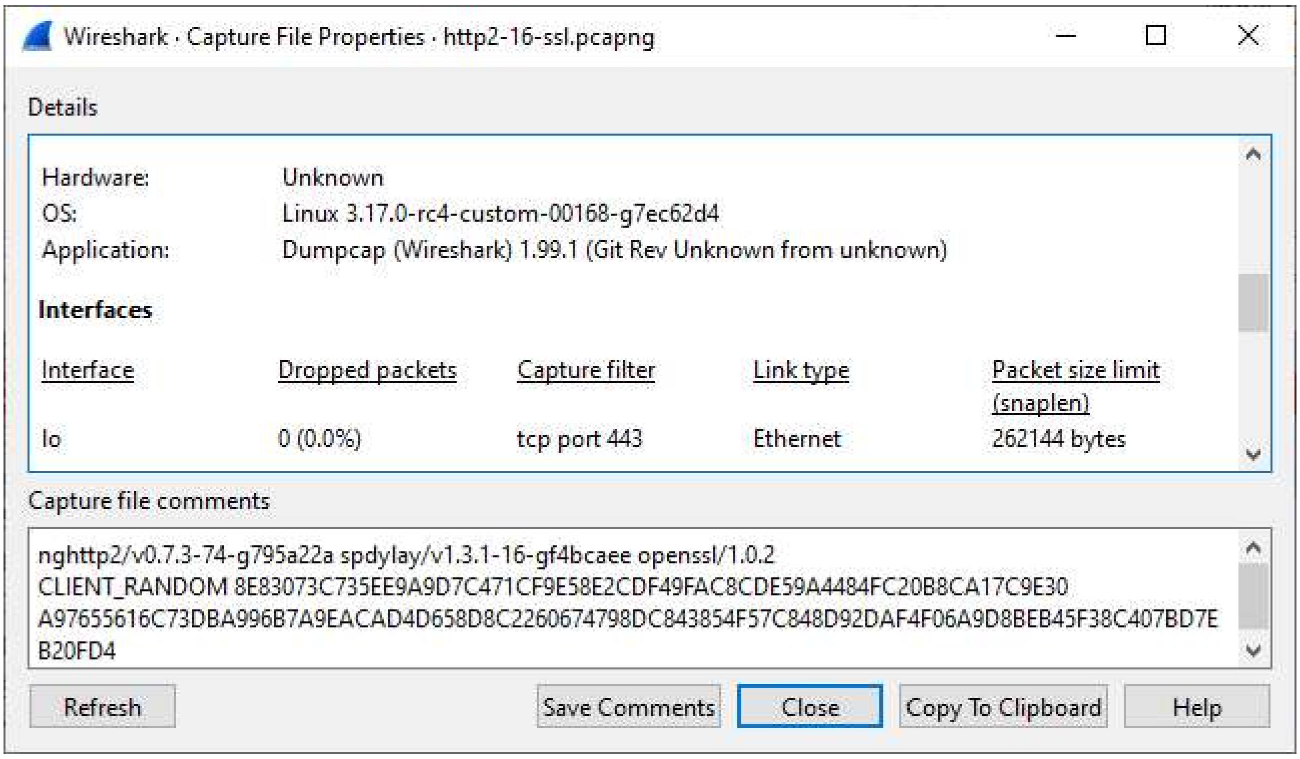
Metadata Block Types

PcapNG files allow a great deal of metadata about the captured packets to be stored, both as optional fields within EPB’s but also in specific block types designed for storing metadata, such as:

- Interface Description Block: network interface name, IP address and MAC address of the device used to capture the traffic.
- Name Resolution Block: maps hostnames to one or several IP addresses or MAC addresses.
- Interface Statistics Block: number of captured and dropped packets
- Decryption Secrets Block: encryption keys used for TLS, WireGuard or ZigBee.

The mandatory Section Header Block also typically contains information about the hardware, operating system and sniffer application used to capture the traffic.

Having all this metadata in the capture files significantly increases the risk of users unwittingly revealing sensitive information, such as internal hostnames, operating systems, used application, usernames, GPS coordinates and even encryption keys when sharing a PcapNG capture file. Metadata in a PcapNG file has, for example, been used to identify a person who had anonymously shared a capture file, which showed that traffic to GitHub was being MITM’ed in China.



Hardware:	Unknown
OS:	Linux 3.17.0-rc4-custom-00168-g7ec62d4
Application:	Dumpcap (Wireshark) 1.99.1 (Git Rev Unknown from unknown)

Interface	Dropped packets	Capture filter	Link type	Packet size limit (snaplen)
lo	0 (0.0%)	tcp port 443	Ethernet	262144 bytes

```
nghttp2/v0.7.3-74-g795a22a spdylay/v1.3.1-16-gf4bcaee openssl/1.0.2
CLIENT_RANDOM 8E83073C735EE9A9D7C471CF9E58E2CDF49FAC8CDE59A4484FC20B8CA17C9E30
A97655616C73DBA996B7A9EACAD4D658D8C2260674798DC843854F57C848D92DAF4F06A9D8BEB45F38C407BD7E
B20FD4
```

Wireshark’s “Capture File Properties” feature can show some of the metadata in a PcapNG file, but not all of it. It is therefore difficult to know what metadata a PcapNG file contains.

So if you need to share captured packets with others, then my recommendation is to use the traditional PCAP (aka libpcap) file format, unless you actually want to share metadata that is only available when using the PcapNG format.

Endianness and Backwards Navigation

All block types in a PcapNG file, both packet block types as well as metadata block types, use this common structure.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Block Type                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Block Total Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                          Block Body                           /
/              variable length, padded to 32 bits               /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Block Total Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Block Type: A 32-bit number indicating if the type of the block (for example 0x00000004 means "Name Resolution Block" and 0x00000006 means "Enhanced Packet Block")
- Block Total Length: The total length of the block
- Block Body: Data depending on block type
- Block Total Length: Trailing block length. Should hold the same length value as specified before Block Body

As you can see, all blocks have a **redundant length field** at the end. These trailing length fields are there in order to facilitate backwards navigation in PcapNG files. One use-case for the backwards navigation is to calculate the capture duration of a PcapNG file by only reading the timestamps of the first and last packets in the capture file. The last packet can often be read with help of the trailing block length like this:

1. Jump to the end of the file
2. Read the 32-bit trailing length field right before the current position
3. Step back $length number of bytes
4. Check if the packet type is 6 (EPB). If not, GOTO #2, otherwise continue
5. Read the timestamp field of the EPB packet (offset 12)

Unfortunately I have to strongly advise against using the trailing block length for backwards navigation because the endianness of the trailing length field is typically unknown when reading backwards. The first block in a PcapNG file is by definition always a SHB, which defines the endianness (big or little endian) for all block field values in that section, but not for the whole capture file. A PcapNG file can contain multiple SHB's, and any one of them can change the endianness.

I find it quite peculiar that the PcapNG file specification supports both big and little endian blocks. It's probably a legacy from the old PCAP format, in which the endianness was defined by the file type magic, where "d4 c3 b2 a1" implies little endian and "a1 b2 c3 d4" means big endian. Most PCAP files use the little endian format, which is why some tools don't even support big endian PCAP files. Allowing capture tools to use the endianness of the local CPU and OS when capturing packets to a file maybe made sense in the 80's, when the PCAP format was designed. But flipping the endianness only requires a single CPU-instruction nowadays, which is just a fraction of the time it takes to retrieve the current time (not to mention slower operations like reading packet data from a network interface or writing a captured packet to disk). It would therefore make sense to simplify the capture file format, and get rid of endian related implementation flaws and incompatibility issues, by specifying a fixed endian to be used in all capture files. Unfortunately that opportunity wasn't capitalized on when the PcapNG file format was created, instead the problem got worse since the PcapNG spec allows for mixed endianness within a single capture file.

32 bit Boundaries and Padding

All blocks in a PcapNG file must be aligned to a 32 bit boundary. This means that padding must be applied to blocks carrying packet data unless the length of the captured data is a multiple of 4 bytes. Padding must also be applied to variable length fields in metadata blocks, such as DNS records in NRB and encryption keys in DSB as well as variable length option fields.

The padding used in PcapNG is called "alignment octets" and is typically appended to variable length fields to make sure they are a multiple of 4 bytes in length. The actual padding should contain only zeros.

My Personal Verdict on PcapNG

I use PcapNG files almost every day and I have written code that both reads and writes PcapNG files, so it's not strange that I sometimes get annoyed by some of the odd quirks in the PcapNG spec. But overall I'm actually happy that we have something like the PcapNG file format, since it enables coherent data to be stored in a single file rather than having to use multiple files. Nevertheless, I try to adhere to the principle of using the PCAP format as default and only use PcapNG when I need to include some type of data that isn't supported by the original PCAP format. As an example, CapLoader automatically generates a capture file for the flows that are currently selected in the GUI. This capture file is typically a PCAP file, but if the selected flows have different link layer types, then the packets get written to a PcapNG file instead.

PcapNG Pros:

- Support for using multiple link layer types in one capture file.
- All block types use the same generic structure.
- Thorough documentation and IETF RFC draft.
- File format is supported by Wireshark and tcpdump.

- Extensible block structure allowing GPS coordinates, running processes, systemd journal entries and other custom data types to be stored.

PcapNG Cons:

- Not as widely supported as the original PCAP format.
- The extra metadata might lead to privacy leaks when sharing capture files.
- Overly complex specification with several rarely used block types and options increase risk of implementation flaws.
- Unnecessary overhead introduced, with features like redundant length fields and 32 bit boundary padding, resulting in a more than doubled per-packet overhead compared to the original PCAP file format.
- Inconsistent representation of 64 bit fields may lead to implementation flaws.
- Timestamps in non-packet blocks risk being parsed incorrectly due to placement of timestamp resolution in IDB.

About the Author

Erik Hjelmvik is well known in the field of network forensics and network monitoring for having created tools such as NetworkMiner, CapLoader, PolarProxy and RawCap as well as for teaching classes in network forensics. Erik added support for the PcapNG file format to NetworkMiner Professional and CapLoader in 2013 and has reported several issues in the PcapNG file specification.