NAME

rawshark − Dump and analyze raw pcap data

SYNOPSIS

rawshark [**−d** <encap:linktype>|<proto:protoname>] [**−F** <field to display>] [**−h**] [**−l**] [**−m** <bytes>] [**−n**] [**−N** <name resolving flags>] [**−o** <preference setting>] ... [**−p**] [**−r** <pipe>|−] [**−R** <read (display) filter>] [**−s**] [**−S** <field format>] [**−t** a|ad|adoy|d|dd|e|r|u|ud|udoy] [**−v**]

DESCRIPTION

Rawshark reads a stream of packets from a file or pipe, and prints a line describing its output, followed by a set of matching fields for each packet on stdout.

INPUT

Unlike **TShark**, **Rawshark** makes no assumptions about encapsulation or input. The **−d** and **−r** flags must be specified in order for it to run. One or more **−F** flags should be specified in order for the output to be useful. The other flags listed above follow the same conventions as **Wireshark** and **TShark**.

Rawshark expects input records with the following format by default. This matches the format of the packet header and packet data in a pcap−formatted file on disk.

```
struct rawshark_rec_s {
    uint32_t ts_sec;      /* Time stamp (seconds) */
    uint32_t ts_usec;     /* Time stamp (microseconds) */
    uint32_t caplen;      /* Length of the packet buffer */
    uint32_t len;         /* "On the wire" length of the packet */
    uint8_t data[caplen]; /* Packet data */
};
```

If **−p** is supplied **rawshark** expects the following format. This matches the *struct pcap_pkthdr* structure and packet data used in libpcap, Npcap, or WinPcap. This structure’s format is platform−dependent; the size of the *tv_sec* field in the *struct timeval* structure could be 32 bits or 64 bits. For **rawshark** to work, the layout of the structure in the input must match the layout of the structure in **rawshark**. Note that this format will probably be the same as the previous format if **rawshark** is a 32−bit program, but will not necessarily be the same if **rawshark** is a 64−bit program.

```
struct rawshark_rec_s {
    struct timeval ts;    /* Time stamp */
    uint32_t caplen;      /* Length of the packet buffer */
    uint32_t len;         /* "On the wire" length of the packet */
    uint8_t data[caplen]; /* Packet data */
};
```

In either case, the endianness (byte ordering) of each integer must match the system on which **rawshark** is running.

OUTPUT

If one or more fields are specified via the **−F** flag, **Rawshark** prints the number, field type, and display format for each field on the first line as "packet number" 0. For each record, the packet number, matching fields, and a "1" or "0" are printed to indicate if the field matched any supplied display filter. A "−" is used to signal the end of a field description and at the end of each packet line. For example, the flags **−F ip.src −F dns.qry.type** might generate the following output:

```
0 FT_IPv4 BASE_NONE − 1 FT_UINT16 BASE_HEX −
1 1="1" 0="192.168.77.10" 1 −
2 1="1" 0="192.168.77.250" 1 −
3 0="192.168.77.10" 1 −
4 0="74.125.19.104" 1 −
```

Note that packets 1 and 2 are DNS queries, and 3 and 4 are not. Adding **−R "not dns"** still prints each line, but there's an indication that packets 1 and 2 didn't pass the filter:

```
0 FT_IPv4 BASE_NONE − 1 FT_UINT16 BASE_HEX −
1 1="1" 0="192.168.77.10" 0 −
2 1="1" 0="192.168.77.250" 0 −
3 0="192.168.77.10" 1 −
4 0="74.125.19.104" 1 −
```

Also note that the output may be in any order, and that multiple matching fields might be displayed.

OPTIONS

−d <encapsulation>

Specify how the packet data should be dissected. The encapsulation is of the form *type:value*, where *type* is one of:

encap:*name* Packet data should be dissected using the libpcap/Npcap/WinPcap data link type (DLT) *name*, e.g. **encap:EN10MB** for Ethernet. Names are converted using pcap_datalink_name_to_val(). A complete list of DLTs can be found at https://www.tcpdump.org/linktypes.html.

encap:*number* Packet data should be dissected using the libpcap/Npcap/WinPcap LINKTYPE_ *number*, e.g. **encap:105** for raw IEEE 802.11 or **encap:101** for raw IP.

proto:*protocol* Packet data should be passed to the specified Wireshark protocol dissector, e.g. **proto:http** for HTTP data.

−F <field to display>

Add the matching field to the output. Fields are any valid display filter field. More than one **−F** flag may be specified, and each field can match multiple times in a given packet. A single field may be specified per **−F** flag. If you want to apply a display filter, use the **−R** flag.

−h

Print the version and options and exits.

−l

Flush the standard output after the information for each packet is printed. (This is not, strictly speaking, line−buffered if **−V** was specified; however, it is the same as line−buffered if **−V** wasn't specified, as only one line is printed for each packet, and, as **−l** is normally used when piping a live capture to a program or script, so that output for a packet shows up as soon as the packet is seen and dissected, it should work just as well as true line−buffering. We do this as a workaround for a deficiency in the Microsoft Visual C++ C library.)

This may be useful when piping the output of **TShark** to another program, as it means that the program to which the output is piped will see the dissected data for a packet as soon as **TShark** sees the packet and generates that output, rather than seeing it only when the standard output buffer containing that data fills up.

−m <memory limit bytes>

Limit rawshark's memory usage to the specified number of bytes. POSIX (non−Windows) only.

−n

Disable network object name resolution (such as hostname, TCP and UDP port names), the **−N** flag might override this one.

−N <name resolving flags>

Turn on name resolving only for particular types of addresses and port numbers, with name resolving for other types of addresses and port numbers turned off. This flag overrides **−n** if both **−N** and **−n** are present. If both **−N** and **−n** flags are not present, all name resolutions are turned on.

The argument is a string that may contain the letters:

m to enable MAC address resolution

n to enable network address resolution

N to enable using external resolvers (e.g., DNS) for network address resolution

t to enable transport−layer port number resolution

d to enable resolution from captured DNS packets

v to enable VLAN IDs to names resolution

−o <preference>:<value>

Set a preference value, overriding the default value and any value read from a preference file. The argument to the option is a string of the form *prefname:value*, where *prefname* is the name of the preference (which is the same name that would appear in the preference file), and *value* is the value to which it should be set.

−p

Assume that packet data is preceded by a pcap_pkthdr struct as defined in pcap.h. On some systems the size of the timestamp data will be different from the data written to disk. On other systems they are identical and this flag has no effect.

−r <pipe>|−

Read packet data from *input source*. It can be either the name of a FIFO (named pipe) or ‘‘−" to read data from the standard input, and must have the record format specified above.

If you are sending data to rawshark from a parent process on Windows you should not close rawshark’s standard input handle prematurely, otherwise the C runtime might trigger an exception.

−R <read (display) filter>

Cause the specified filter (which uses the syntax of read/display filters, rather than that of capture filters) to be applied before printing the output.

−s

Allows standard pcap files to be used as input, by skipping over the 24 byte pcap file header.

−S

Use the specified format string to print each field. The following formats are supported:

%D Field name or description, e.g. "Type" for dns.qry.type

%N Base 10 numeric value of the field.

%S String value of the field.

For something similar to Wireshark’s standard display ("Type: A (1)") you could use **%D: %S (%N)**.

−t a|ad|adoy|d|dd|e|r|u|ud|udoy

Set the format of the packet timestamp printed in summary lines. The format can be one of:

a absolute: The absolute time, as local time in your time zone, is the actual time the packet was captured, with no date displayed

ad absolute with date: The absolute date, displayed as YYYY−MM−DD, and time, as local time in your time zone, is the actual time and date the packet was captured

adoy absolute with date using day of year: The absolute date, displayed as YYYY/DOY, and time, as local time in your time zone, is the actual time and date the packet was captured

d delta: The delta time is the time since the previous packet was captured

dd delta_displayed: The delta_displayed time is the time since the previous displayed packet was captured

e epoch: The time in seconds since epoch (Jan 1, 1970 00:00:00)

r relative: The relative time is the time elapsed between the first packet and the current packet

u UTC: The absolute time, as UTC, is the actual time the packet was captured, with no date displayed

ud UTC with date: The absolute date, displayed as YYYY−MM−DD, and time, as UTC, is the actual time and date the packet was captured

udoy UTC with date using day of year: The absolute date, displayed as YYYY/DOY, and time, as UTC, is the actual time and date the packet was captured

The default format is relative.

−v

Print the version and exit.

READ FILTER SYNTAX

For a complete table of protocol and protocol fields that are filterable in **TShark** see the wireshark−filter(4) manual page.

FILES

These files contains various **Wireshark** configuration values.

Preferences

The *preferences* files contain global (system−wide) and personal preference settings. If the system−wide preference file exists, it is read first, overriding the default settings. If the personal preferences file exists, it is read next, overriding any previous values. Note: If the command line option **−o** is used (possibly more than once), it will in turn override values from the preferences files.

The preferences settings are in the form *prefname:value*, one per line, where *prefname* is the name of the preference and *value* is the value to which it should be set; white space is allowed between **:** and *value*. A preference setting can be continued on subsequent lines by indenting the continuation lines with white space. A **#** character starts a comment that runs to the end of the line:

```
# Capture in promiscuous mode?
# TRUE or FALSE (case−insensitive).
capture.prom_mode: TRUE
```

The global preferences file is looked for in the *wireshark* directory under the *share* subdirectory of the main installation directory (for example, */usr/local/share/wireshark/preferences*) on UNIX−compatible systems, and in the main installation directory (for example, *C:\Program Files\Wireshark\preferences*) on Windows systems.

The personal preferences file is looked for in *$XDG_CONFIG_HOME/wireshark/preferences* (or, if *$XDG_CONFIG_HOME/wireshark* does not exist while *$HOME/.wireshark* is present, *$HOME/.wireshark/preferences*) on UNIX−compatible systems and *%APPDATA%\Wireshark\preferences* (or, if %APPDATA% isn't defined, *%USERPROFILE%\Application Data\Wireshark\preferences*) on Windows systems.

Disabled (Enabled) Protocols

The *disabled_protos* files contain system−wide and personal lists of protocols that have been disabled, so that their dissectors are never called. The files contain protocol names, one per line, where the protocol name is the same name that would be used in a display filter for the protocol:

```
http
tcp     # a comment
```

The global *disabled_protos* file uses the same directory as the global preferences file.

The personal *disabled_protos* file uses the same directory as the personal preferences file.

Name Resolution (hosts)

If the personal *hosts* file exists, it is used to resolve IPv4 and IPv6 addresses before any other attempts are made to resolve them. The file has the standard *hosts* file syntax; each line contains one IP address and name, separated by whitespace. The same directory as for the personal preferences file is used.

Capture filter name resolution is handled by libpcap on UNIX−compatible systems and Npcap or WinPcap on Windows. As such the Wireshark personal *hosts* file will not be consulted for capture filter name resolution.

Name Resolution (subnets)

If an IPv4 address cannot be translated via name resolution (no exact match is found) then a partial match is attempted via the *subnets* file.

Each line of this file consists of an IPv4 address, a subnet mask length separated only by a / and a name separated by whitespace. While the address must be a full IPv4 address, any values beyond the mask length are subsequently ignored.

An example is:

# Comments must be prepended by the # sign! 192.168.0.0/24 ws_test_network

A partially matched name will be printed as "subnet−name.remaining−address". For example, "192.168.0.1" under the subnet above would be printed as "ws_test_network.1"; if the mask length above had been 16 rather than 24, the printed address would be ‘‘ws_test_network.0.1".

Name Resolution (ethers)

The *ethers* files are consulted to correlate 6−byte hardware addresses to names. First the personal *ethers* file is tried and if an address is not found there the global *ethers* file is tried next.

Each line contains one hardware address and name, separated by whitespace. The digits of the hardware address are separated by colons (:), dashes (−) or periods (.). The same separator character must be used consistently in an address. The following three lines are valid lines of an *ethers* file:

```
ff:ff:ff:ff:ff:ff          Broadcast
c0−00−ff−ff−ff−ff          TR_broadcast
00.00.00.00.00.00          Zero_broadcast
```

The global *ethers* file is looked for in the */etc* directory on UNIX−compatible systems, and in the main installation directory (for example, *C:\Program Files\Wireshark*) on Windows systems.

The personal *ethers* file is looked for in the same directory as the personal preferences file.

Capture filter name resolution is handled by libpcap on UNIX−compatible systems and Npcap or WinPcap on Windows. As such the Wireshark personal *ethers* file will not be consulted for capture filter name resolution.

Name Resolution (manuf)

The *manuf* file is used to match the 3−byte vendor portion of a 6−byte hardware address with the manufacturer’s name; it can also contain well−known MAC addresses and address ranges specified with a netmask. The format of the file is the same as the *ethers* files, except that entries of the form:

```
00:00:0C      Cisco
```

can be provided, with the 3−byte OUI and the name for a vendor, and entries such as:

```
00−00−0C−07−AC/40     All−HSRP−routers
```

can be specified, with a MAC address and a mask indicating how many bits of the address must match. The above entry, for example, has 40 significant bits, or 5 bytes, and would match addresses from 00−00−0C−07−AC−00 through 00−00−0C−07−AC−FF. The mask need not be a multiple of 8.

The *manuf* file is looked for in the same directory as the global preferences file.

Name Resolution (services)

The *services* file is used to translate port numbers into names.

The file has the standard *services* file syntax; each line contains one (service) name and one transport identifier separated by white space. The transport identifier includes one port number and one transport protocol name (typically tcp, udp, or sctp) separated by a /.

An example is:

```
mydns       5045/udp     # My own Domain Name Server
mydns       5045/tcp     # My own Domain Name Server
```

Name Resolution (ipxnets)

The *ipxnets* files are used to correlate 4−byte IPX network numbers to names. First the global *ipxnets* file is tried and if that address is not found there the personal one is tried next.

The format is the same as the *ethers* file, except that each address is four bytes instead of six. Additionally, the address can be represented as a single hexadecimal number, as is more common in the IPX world, rather than four hex octets. For example, these four lines are valid lines of an *ipxnets* file:

```
C0.A8.2C.00                 HR
c0−a8−1c−00                 CEO
00:00:BE:EF                 IT_Server1
110f                        FileServer3
```

The global *ipxnets* file is looked for in the */etc* directory on UNIX−compatible systems, and in the main installation directory (for example, *C:\Program Files\Wireshark*) on Windows systems.

The personal *ipxnets* file is looked for in the same directory as the personal preferences file.

ENVIRONMENT VARIABLES

WIRESHARK_CONFIG_DIR

This environment variable overrides the location of personal configuration files. It defaults to *$XDG_CONFIG_HOME/wireshark* (or *$HOME/.wireshark* if the former is missing while the latter exists). On Windows, *%APPDATA%\Wireshark* is used instead. Available since Wireshark 3.0.

WIRESHARK_DEBUG_WMEM_OVERRIDE

Setting this environment variable forces the wmem framework to use the specified allocator backend for **all** allocations, regardless of which backend is normally specified by the code. This is mainly useful to developers when testing or debugging. See *README.wmem* in the source distribution for details.

WIRESHARK_RUN_FROM_BUILD_DIRECTORY

This environment variable causes the plugins and other data files to be loaded from the build directory (where the program was compiled) rather than from the standard locations. It has no effect when the program in question is running with root (or setuid) permissions on *NIX.

WIRESHARK_DATA_DIR

This environment variable causes the various data files to be loaded from a directory other than the standard locations. It has no effect when the program in question is running with root (or setuid) permissions on *NIX.

ERF_RECORDS_TO_CHECK

This environment variable controls the number of ERF records checked when deciding if a file really is in the ERF format. Setting this environment variable a number higher than the default (20) would make false positives less likely.

IPFIX_RECORDS_TO_CHECK

This environment variable controls the number of IPFIX records checked when deciding if a file really is in the IPFIX format. Setting this environment variable a number higher than the default (20) would make false positives less likely.

WIRESHARK_ABORT_ON_DISSECTOR_BUG

If this environment variable is set, **Rawshark** will call abort(3) when a dissector bug is encountered. abort(3) will cause the program to exit abnormally; if you are running **Rawshark** in a debugger, it should halt in the debugger and allow inspection of the process, and, if you are not running it in a debugger, it will, on some OSes, assuming your environment is configured correctly, generate a core dump file. This can be useful to developers attempting to troubleshoot a problem with a protocol dissector.

WIRESHARK_ABORT_ON_TOO_MANY_ITEMS

If this environment variable is set, **Rawshark** will call abort(3) if a dissector tries to add too many items to a tree (generally this is an indication of the dissector not breaking out of a loop soon enough). abort(3) will cause the program to exit abnormally; if you are running **Rawshark** in a debugger, it should halt in the debugger and allow inspection of the process, and, if you are not running it in a debugger, it will, on some OSes, assuming your environment is configured correctly, generate a core dump file. This can be useful to developers attempting to troubleshoot a problem with a protocol dissector.

SEE ALSO

wireshark−filter(4), wireshark(1), tshark(1), editcap(1), pcap(3), dumpcap(1), text2pcap(1), pcap−filter(7) or tcpdump(8)

NOTES

This is the manual page for **Rawshark** 3.6.2. **Rawshark** is part of the **Wireshark** distribution. The latest version of **Wireshark** can be found at https://www.wireshark.org.

HTML versions of the Wireshark project man pages are available at https://www.wireshark.org/docs/man−pages.

AUTHORS

Rawshark uses the same packet dissection code that **Wireshark** does, as well as using many other modules from **Wireshark**; see the list of authors in the **Wireshark** man page for a list of authors of that code.