AnetTest utility

User guide

Contents

1. General description

The program Anettest may be used as a packet generator. It is possible to generate any packets at channel layer (for Ethernet networks), some types of packets at network layer (IP protocol), and at application layer (sending data blocks within TCP session).

Descriptions of packets can be stored in text files or be specified in command line. The syntax of description is simple enough (field name - value). The fields (a format of packet) are specified in header files (headers). There are headers provided for commonly known network protocols. The program allows you to define own headers. Numerical data types, strings, IP address, MAC addresses are available.

The generation of packets can be alternated with waiting of others. In the case of waiting a packet mask is used which is based on packet's content (an equality condition) but it may also have special conditions: not equal, greater than, etc.

The program provides a simple but rather flexible language with variable and branches. Thus, the statuses of received packets, their content may be deeply analyzed to decide on following actions.

It is possible to use the program for implementing the automated tests of various network devices and applications. It is achieved by the ability that after packet's description you may specify the requests according to which the packets will be searched in trace-files or network will be scanned in real time. Generally it will be possible to generate sets of packets, then wait the reaction of various network devices and applications caused by these packets and then in the total obtain the message "successful test" or the list of discrepancies.

It is possible to use the program as a slightly unusual sniffer. If you describe a set of packets in a file the program may be a scanner of network about the presence of specified packets. At a specified interval a report is displayed about the total number and the intensity of packets of each type.

The program also has several operation modes for network filter (firewall) testing and the ability of modifying trace-files.

2. General notions

Main interface - by default it's the first interface specified by option -d. From this interface packets will be generated. For some commands just on this interface packets will be waited by default (command WAIT). It can be changed by command MI.

Packet mask – a part of packet's description which will be used while comparison the packet with anyone accepted on physical interface or from trace-file. In simple case the mask consists of those fields for which values have been specified in packet's description but it is not always true (see command RESET, special mask conditions). Only values of fields from mask will be considered while comparison a packet with another. More about packet mask…

Reception interfaces – physical network interfaces of computer or trace-files, i.e. packets can be captured in real time, or obtained from specified trace-files. The logic of work does not change. Requests to packet (as parameters to command SEND or WAIT) will correspond to each reception interface. For the most of operating modes reception interfaces – all opened physical interfaces (will go in the same order as they will be specified by option -d). In online-test of packet filter the first opened physical interface becomes the main interface, the others will be reception interfaces.

Requests to packet – the list of requests like "is accepted" (ACCEPT, >>), "is not accepted" (DROP, <<), «has no meaning» (ANY). Can be specified for any packet. Each request corresponds to one reception interface. The first in the list - for the first reception interface, the second - for the second, etc. They are specified as parameters to commands SEND and WAIT.

Syntax of the list:

```
<request>:: = “<<” | “>>” | “DROP” | “ACCEPT” | “ANY”
<list of requests>:: = { <request> }
```

While working in extended mode the list of requests is another.

User number of interface – the number used as the name of interface instead of its system name. The user number of physical interface may be specified while its opening by –d option. The used number of trace-file may be specified while its opening by –c option.

3. Basic operating modes

All options and work modes are described in Table 1.

Further the review of the basic modes with examples is resulted.

3.1. Packet generator

For the generation of packets start the program by one of the next ways:

```
anettest -d eth0 -f packets.fws
anettest -d 10.0.0.1 -f packets.fws
```

By means of option -d the name of a network adapter for work at channel level may be specified (ex: *eth0*). It is possible to look at the list of accessible adapters, using option -i. Instead of adapter's name it is possible to specify the corresponding IP-address (as shown in the second case). The IP-addresses corresponding to adapters can also be obtained by option -i.

The description of packets can be stored in text file which may be specified by option -f.

File *packets.fws* (the test script) can be like the following:

```
INCLUDE tcp.fws
srcip lpc10
dstip 1.1.1.1
dstport 80
        SEND
```

Before packet's description it is recommended to include one of header files using command INCLUDE. Instead of "*include tcp.fws*" it would be possible to type simply "tcp", and program would try to find a similar header file. The header file "tcp.fws" includes other header files so, that values of all the fields of each network header (Ethernet, IP, TCP) are initialized by correct default values, otherwise these values would just be equal to 0. In the shown example after the inclusion of header file the values of three fields (with names *srcip*, *dstip*, *dstport*) are changing how it is necessary for concrete packet. Command SEND generates the packet described before it. After command SEND it would be possible to continue the packet description, i.e. change values of some fields, and then use command SEND again to send the second packet which is different from the first. More about the syntax of script.

In the program distribute there must be a HEADERS folder with headers of network protocols. Names of fields of network protocols can be obtained by watching corresponding header files or by using option -k.

It is possible to specify packet's description directly in command line:

```
anettest -d eth0 tcp dstip 1.1.1.1 dstport 80
```

The feature: in the end of packet's description command SEND is automatically added so it's not necessary to type it. Word "tcp" means including of header file (this word is treated as a part of packet's description in command line).

Ex: samples/generate_udp_packet.fws, samples/generate_sequence_packets.fws.

See also:

options -T (working with data segments over TCP-sessions),

command REP (multiple generation),

command PAUSE,

command INC (autoincrementing value of some field).

3.2. Tracing packets

The program implements one of standard functions of sniffers - tracing packets and recording to file. The program does not display packet's description but only writes down their content in file (a file format libpcap). The physical adapter (where to trace packets) is specified by option -d. The name of file for record is specified by option -t. Packets content are fully stored by default (see also option -s).

Example:

```
anettest -d eth0 -t trace.pcap
```

Note: if the intensity of packets will be too high during enough big time interval then some packets might not be registered (are not written down in file). Nevertheless, the general number of packets accepted on the main interface which will be displayed at the end of program work, will show the real number of accepted packets.

```
Total number of packets recieved on first interface = 1
```

3.3. Packet filter test

It is a mode for implementing automated tests of various packet filters. Automatization means that you don't need for manual review of sniffer's output in order to conclude if packets are passed through filter.

If the all used network segments (corresponding to physical interfaces) where it is necessary to trace packets, are connected to one computer then it's possible to implement fast test: while generating packets tracing will be performed. Right after generation of all packets the result of test will be available (called *online-test*). If network segments are connected to different hosts then while packets generating from one host it is necessary to trace packets on other hosts, recording them in trace-files and then to copy all of these files on one computer and run *offline-test*. Both tests will be described below.

The file with the description of packets which is set through an option-f, can look so:

```
fasttest // while online-test this command must be specified at the beginning
      of script

INCLUDE tcp
fullmask
tcp.flags = syn
dstport http
    send drop accept
dstport ssh
    send accept drop
dstport ftp
    send drop accept
```

Here three packets are described. They are separated from each other by the command SEND. The packets differ in the values of destination TCP-port (dstport). The values of TCP-flags (tcp.flags) are equal for all the packets. The values of other fields are defined in the included header file tcp. After the each command SEND requests for packet (ACCEPT or DROP) are specified. More details about syntax of script are in other section.

The using of FULLMASK will be explained below.

The Offline-test

To implement the offline-test at first it's necessary to start sniffers on all participating hosts, for example:

```
anettest -d eth0 -t trace1.fws
anettest -d eth0 -t trace2.fws
anettest -d eth0 -t trace3.fws
```

These sniffers will store any received packets in file.

Then start packet generator on one computer, for example:

```
anettest -f packets.fws -d eth0
```

Then collect all the resulted trace-files in one place and perform the direct offline-test:

```
anettest -f packets.fws -c trace1 -c trace2 -c trace3
```

During the direct offline-test each of the packets described in the script "packets.fws" will be searched in the trace-files. While the comparison of packets **packet mask** will be used. The result of search is compared with the requests to packets which have been mentioned earlier. The first request after command SEND will correspond to the file "trace1", the second and the third - to the files "trace2" and "trace3". ACCEPT - means that the packet must be found in a trace-file, DROP – not found. Thus, the program checks the conformance of the requests specified in the file "packets.fws", and search results for the trace-files. The message "successful test" that all requests are met, or the list of discrepancies will be

displayed.

The offline-test mode is always turned on by option -c.

The Online-test

Trace-file is a special type of reception interface. Files trace1, trace2, trace3 were the first, second and third reception interfaces.

If all of reception interfaces are connected to one host then it's possible to perform fast online-test:

Only one start is necessary:

```
anettest -f packets.fws -d eth1#1 -d eth2#2 -d eth3#3
```

After symbol # user number of the interface is specified.

For implementing such a test at the beginning of script file command FASTTEST must be typed which turns on the online-test mode. If not to use this command packets from script will be simply generated.

Packets will be generated from interface eth1 (main interface) and will be registered on reception interfaces: eth2, eth3. The first request will correspond to eth2, the second - eth3. In other operating modes (sniffer mode, using command WAIT) all open physical interfaces are reception interfaces, i.e. the first requirement would correspond to eth1. While online-test only there is a shift.

During the test at first sniffers will be started on all opened interfaces, then there will be an instant generation of all packets from script from the main interface. After the generation of packets sniffers will be stopped, and there will be performed the search of packets from script among the packets registered by sniffers (is similarly to searching in trace-files).

Pause between starting of sniffers and the beginning of generation of packets = 100 ms.

Pause between the termination of generation and sniffers termination = 200 ms.

If packets in the beginning or in the end of script are registered as not accepted (though actually they are passed through filter either too quickly, or too slowly) try to increase pauses by command PAUSE, having inserted it in the beginning or in the end.

Reports

The output of program for any type of test may be like this:

```
Packet on line 10 (./test.fws): accepted (dev 2)
Packet on line 14 (./test.fws): droped (dev 2)
Packet on line 19 (./test.fws): accepted (dev 2)
```

For each found discrepancy the name of file and the line in file are displayed, where packet's description was met (more precisely the position of command SEND before which there was packet's description), the result of search for the concrete reception interface (interface number is enclosed in brackets). If user number has been specified for interface (in option –d) then it will be shown (with prefix "dev"), otherwise the consecutive number according to the order of opening interfaces will be shown (with prefix "sdev").

Command NAME allows to set the name for packet which will be displayed in report.

Command REP allows to specify that must be received not one but several packets.

Remark:

If in the previous example someone not used the command FULLMASK then while the comparison of packets there would be considered only those fields whose values are explicitly specified in the script

(they are *tcp.flags* and *dstport*). The other fields of tcp header whose values are specified in the file “tcp.fws” would not participate in the comparison because the command RESET is used in standard header files including tcp header (it has been done for convenience while working in other operating modes). Thus, it would be possible not to notice some changes of packet’s content on their way or to treat foreign tcp packets as own ones.

Examples: samples/compare_mode.fws, samples/compare_mode1.fws, samples/fasttest.fws.

3.4. Extended mode

The extended mode is necessary, when packets during the test should be sent from different interfaces.

This mode is turned on by command EXTENDED.

In this mode main interface can be changed (command MI). Thus, in one script it is possible to generate packets from different physical interfaces.

Requests for packets in this mode should be specified differently:

```
<request1>:: = “DROP” | “ACCEPT” | “ANY”
<request2>:: = “<<” | “>>”
<request>:: = (<request1> “ “ <user number of interface>) | (<request2> [“ “]
      <user number of interface>)
<list of requests>:: = { <request> }
```

That is after each request there should be the user number of interface for which request is specified. << and >> may not be separated by space from user number.

Example:

```
SEND <<0 >>1 <<2
SEND DROP 0 ACCEPT 1 DROP 2
```

Similarly requests in command DEFAULTS and in command WAIT must be specified.

3.5. Alternation of sending and receiving packets

Alternation of sending and receiving packets can be implemented in general operating mode. Sent and waited packets can be different. The test can be stretched in time by command PAUSE. It is possible to choose the moments of the beginning of waiting of some packet.

The program should be started in usual mode (as packet generator):

```
anettest -d eth0 -f packets.fws
```

To generate packets it is possible to use command SEND (without parameters).

To wait packets use WAIT, OR, WAITALL. It is possible to specify a timeout of waiting by command TIMEOUT.

Results of waiting of packets that has been added by commands WAIT or OR will be displayed in report which will be displayed while program termination. Command CLEARREG typed in the ending of script causes that report will not be displayed.

3.6. Sniffer mode

The sniffer will display statistics for packets described in script (separated by SEND command), show their intensity (the number of packets and the amount of data) in real time, displaying periodically reports. This mode is alternative to "packet filter test" since there will be displayed information about the conformance of current state of network and the requests to packets specified in scenario. As well as in the test of the packet filter packets are divided by command SEND after which it is possible to specify requirements to packet.

Example of start of the program:

```
anettest -d eth0 –r -v srcport 22
```

Program will show statistic about packets with number of source port = 22. The sniffer mode is turned on by option -r. The used network adapters are specified by option -d. Packet's description may be specified in command line (like in shown example). It is possible to describe several packets in file (option -f) then the program will periodically display statistic for each of packets in the form of the table (report). While working in this mode not packet's content but packet mask will be used when comparing packet from script with some received packet.

In the previous example to the end of packet's description command SEND will be added (like in the mode of packet generator).

ALL open physical interfaces specified by option -d will be reception interfaces. Unlike the test of packet filter in sniffer mode the first network adapter which is specified by option -d, will be also the first reception interface (i.e. the first request to packet will correspond to it). All subsequent adapters will be considered as the second, the third … reception interfaces. This difference should be considered while using commands SEND or DEFAULTS.

Example: samples/tracing_packets.fws.

See also: option -u (changing period of displaying reports).

3.7. Changing trace files

The program allows to change trace files (having libpcap format) by special commands. Trace file can be open by command TRACE. To write down the changed file on disk there is a command WRITE.

After file opening it is possible to use some simple commands:

getpac - loads packet from trace file in the buffer of currently described packet;

setpac - writes down current packet in file over existing packet;

inspac - writes down current packet in file, inserting a new packet into trace file;

delpac - deletes specified packet from trace file.

There is a command for changing several packets at once: chtrace.

Examples: samples/work_with_traces.fws, samples/work_with_traces1.fws.

3.8. Imitation of network application's work

Imitation of application's work is performed based on information from trace-file (specified by command TRACE). During the test program imitates the work of one or several network applications, sending packets in strict sequence (corresponding to trace file) from different physical interfaces (corresponding to defined test configuration – set of end points).

The program not only sends packets, but also watches, that they have been successfully accepted. In requests to the result of test it is possible to specify, what packets should be accepted, and what are not. The program can send the same packets from one physical interface and wait them on another. Thus it is supposed, that between physical interfaces there is an intermediate point which does not pass some packets (firewall). So you can implement automated tests of application firewalls using prepared templates represented in trace files.

Before imitation start (command RUN) it is necessary to define some end points (command EP, header "configSession"), for example:

```
ep gen 0 srcip first
ep recv 1 srcip first
ep gen 1 srcip second
ep recv 0 srcip second
```

The example corresponds to the imitation of an interaction between two hosts. The first packet in trace file must be from the first host, the second packet – from the second host. The first end point (generating one) indicates that packets from trace file that have the value of field “srcmac” equal to the value from first packet must be generated from the interface with user number = 0. The second end point (receiving one) indicates that the same packets are expected to be received on the interface with user number = 1.

During imitation packets from trace-file are obtained consistently. Each generating end point can send its packet, only when all previous packets are accepted by receiving end points. If some receiving end point waits for its packet too long then all the packets after the last successfully accepted by one of receiving end points will be repeatedly generated. After several repeated generations (NUMRET) the packet will be marked as dropped, and the test will be continued. A certain packet is waited at any moment. When the packet comes (or it is marked as dropped), the transition to the following packet is performed. The packets belonging to one of receiving end points will be waited only on the interface specified for their end point.

Thus, waiting of packets synchronizes the process of their generation. Each packet can belong to a single generating end point and a single receiving one. If receiving end points are not specified, packets will be simply generated. If generating end points are not specified, packets will be simply waited.

It is possible to start two parties of imitator on different hosts. The timeout for the first packet is increased to 3 seconds in order to after the start of waiting have a time to start the generation of packets. If you don’t satisfy with 3 seconds then the first packet may be marked as dropped.

Configured test may be started by command RUN.

The program must be started in general mode:

```
anettest -d eth0#0 -d eth1#1 -f session_test.fws
```

After implementing the first test it is possible to reset the configuration by command DEFAULTTEST and then configure the next test.

If some intermediate point changes packets on their way (in a priory known manner) it is possible to use command RM. It may be useful when testing NAT (Network Address Translation).

If before the test it is not known how the intermediate point will change packets, but it will become known during the test then it is possible to use command ARM (testing NAT).

If the values of some fields of packets do not need to be considered while comparison, then it is possible to use command CIEVE.

Command NUMRET sets the number of repeated generations.

Command TIMEOUT set the timeout before repeated generation.

Example:

Trace-file content:

```
1 0.000000 194.85.99.33 88.210.60.143 TCP 54840> smtp [SYN] Seq=0 Len=0 WS=1
2 0.013321 88.210.60.143 194.85.99.33 TCP smtp> 54840 [SYN, ACK] Seq=0 Ack=1
3 0.013750 194.85.99.33 88.210.60.143 TCP 54840> smtp [ACK] Seq=1 Ack=1
4 0.030554 88.210.60.143 194.85.99.33 SMTP Response: 220 imap.r-and-k.com
      ESMTP
5 0.030996 194.85.99.33 88.210.60.143 TCP 54840> smtp [ACK] Seq=1 Ack=37
6 0.032663 194.85.99.33 88.210.60.143 SMTP Command: EHLO xperts1.rtc.ru
7 0.046934 88.210.60.143 194.85.99.33 SMTP Response: 250-imap.r-and-k.com
```

Test configuration (see command EP):

```
//generating end point 1
ep gen 0 srcip 194.85.99.33       //generating packets for client on
      interface 0
//generating end point 2
ep gen 2 srcip 88.210.60.143      //generating packets for server on
      interface 2
//recieving end point 3
ep recv 2 srcip 194.85.99.33      //waiting packets for client on interface 2
//recieving end point 4
ep recv 0 srcip 88.210.60.143     //waiting packets for server on interface 0
```

The same configuration can be also written so:

```
//generating end point 1
ep gen 0 srcip first              //generating packets for client on
      interface 0
//generating end point 2
ep gen 2 srcip second             //generating packets for server on
      interface 2
//recieving end point 3
ep recv 2 srcip first             //waiting packets for client on interface 2
//recieving end point 4
ep recv 0 srcip second     //waiting packets for server on interface 0
```

At the imitation beginning the packet which must be first waited will be found. It is packet 1 belonging to end point 3. The packet will be waited on interface 2. Then the packet which must be generated will be found. It is packet 1 belonging to end point 1.

Packet 1 will be generated on interface 0. The following packet which needs to be generated is packet 2 belonging to end point 2. This packet cannot be generated, while packet 1 is waited on interface 2. When the packet 1 will be accepted on interface 2 transition to a following waited packet will be performed. It is packet 2 belonging to end point 4. After such a transition packet 2 can be generated by end point 2.

If packet 1 is not accepted during timeout on interface 2 then packet 1 will be repeatedly generated on interface 0. After several repeated generations packet 1 can be marked as droped and waiting of packet 2 will be started.

Full content of test script may be the following:

```
trace smtp_client.pcap

//generating end point 1
ep gen 0 srcip 194.85.99.33       //generating packets for client on
      interface 0
```

```
//generating end point 2
ep gen 2 srcip 88.210.60.143      //generating packets for server on
      interface 2
//recieving end point 3
ep recv 2 srcip 194.85.99.33      //waiting packets for client on interface 2
//recieving end point 4
ep recv 0 srcip 88.210.60.143     //waiting packets for server on interface 0

run drop 6 // all the packets, since 6 should be droped
```

Examples: samples\convtest1.fws, samples\convtest2.fws.

The standard configuration for imitation of client-server session is presented in file headers\configSession.fws.

The standard configuration for imitation of client-server session with NAT between them - headers\natConfigSession.fws.

Configuration NAT should be stored in file headers\natDefines.fws.

4. Command line

Table 1. Command line options

Name	Description	Example
List of adapters (-i)	Displays the list of physical interfaces (adapters) for generating and tracing packets on channel level. In the list after adapter's number there is his name and may be IP address if it has been associated.	`anettest -i`

Opening adapter (-d)	Parameter to option: the name of opened interface.
	By default the adapters for work on channel level are opened. For the list of available adapters use option –i.
	It is possible to specify an adapter by its IP-address or DNS-name.
	Interface's name may contain symbol # after which a user number of adapter is specified.
	When using TCP the format of interface's name is different (see option –T).
		`anettest -d eth0 -f packets.fws`
		
		`anettest -d 10.0.0.10`
		`-d 10.0.1.10`
		`-d 10.0.2.10`
		`-f packets.fws`
		
		`anettest -d 10.0.0.10#0`
		`-d 10.0.1.10#1`
		`-d 10.0.2.10#2`
		`-f packets.fws`

Opening trace-files (-c)	Parameter to option: the name of opened trace file (having libpcap format).
	If this option is specified then program will search packet from script in trace-files (offline-test). See the description of packet filter test.
	Note: if the size of stored part of packet in trace is less than the size of packet from script then packets are not equal.
	If in the name of trace-file there is a number then it will become the user number of interface.
		`anettest -c file1.pcap`
		`-c file2.pcap`
		`-c file3.pcap`
		`-f packets.fws`

Tracing packets to file (-t)	Parameter to option: the name of trace file for recording captured packets. If this option is specified then program will capture packet and record them in trace file (having libpcap format). Packet's content is fully copied by default. Consider the use of option –s.	`anettest -t file.pcap -d eth0`
Test script (-f)	The scenario of work of the program (the description of packets, etc.). In the majority of operating modes it needs to be set. The description of syntax of the scenario is resulted in separate section of the documentation. It is possible to transfer the	`anettest -f packets.fws -d eth0`

	description of packets in a command line. The first word not similar to an option will be considered as the packet description.	
Snaplen (-s)	Parameter to option: the maximum size of packet which will be captured. This parameter is applied globally for all tracing procedures. Implemented only for UNIX.	anettest -t file.pcap -d eth0 –s 120
Sniffer mode (-r)	Activates the sniffer mode.	anettest -r -f packets.fws -d eth0
RAW IP (-p)	Indicates to use generation and tracing packets not on channel level, but on network (IP) level. The names of adapters which are specified in option -d, must be IP addresses. For packets generation only it is not necessary to specify IP address. The Generation of the majority of packets in OS Windows XP will be blocked by system with Service Pack 2 (any TCP packets, packets with no local IP source address). In Linux it is impossible to generate datagramms with size larger than MTU. IP datagramms may be captured only globally without distinguishing interfaces. In FreeBSD neither generation of packets nor tracing are not implemented. For other systems program not tested.	anettest -p …
Working with TCP-sessions (-T)	This option specifies to work not with packets, but with blocks of data over TCP-connection. In option -d it is necessary to specify an adapter name in format client:host:port (client mode) or server:host:port (server mode). Parameter to option: the timeout when waiting for connections while server mode. In client mode program will try to establish a connection to the specified address. In server mode program will wait for connections on the specified address (so the address must be local). In script you should use the field with name tcp.data (or its alias - td) to specify what data must be send (by command SEND) or be waited (by command WAIT). In this mode command WAIT takes the full block of data received from other side and compares it to that packet which is described before command.	anettest –T 99999 –d client:remotehost:110 -f data.fws anettest –T 99999 –d server:localhost:110 -f data.fws anettest -T 9999 -d c:lpc10:21 -f data.fws anettest –T 99999 –d s:localhost:110 -f data.fws

	So the block will be treated as one packet. If at least one byte is received then it will be a packet. So there is no ability to implement absolutely reliable tests but it works in practice. **Note**: while working in this mode command RESET must be used after inclusion of standart headers (ex: tcp): all fields except *tcp.data* must not be in packet mask. See samples/mail_reader.	
-h	Causes that numbers in script will be processed as hexadecimal by default. Without using this option - decimal.	
Verbose output (-v)	If this option is used when in report there will be displayed statistic for all packets. Otherwise – only for those packet for which requests aren't met.	
-w	Causes that program asks for pressing any key before terminating.	
-q	Causes that network adapters are not switched in promisc mode (see libpcap documentation).	
-u	Parameter to option: the interval in milliseconds between displaying reports while working in sniffer mode.	`anettest –u 1000`
Search paths (-I)	Parameter to option: a new path where included files will be searched (see command INCLUDE).	`anettest –headers -I`

Fields info (-k)	Prints the table with information about defined fields of network protocols.
	In command-line you can specify the name of concrete header(s). Program will process this header(s) and then will print the table.
		`anettest -k tcp`
		`anettest -k tcp_header`
		`anettest –k all_headers`

4.1. Program return status

If a fatal error raises then program terminates, returning status = 1.

If there are no fatal errors (test was completed) but specified requests don't correspond with test results then returned status = 2.

If test is successfully completed (all requests correspond with test results) then returned status = 0.

Command EXIT enables you to explicitly define the returned status.

5. Anettest script

Script file is a usual text file which can be edited in any text editor. It is recommended to create such files with *.fws extension (for example: packets.fws).

Comments are the same as for languages C/C ++, Java.

// - the beginning of comments till the end of line.

/* … */ - comments of any number of lines.

Most of elements must be separated by space.

Formal syntax:

```
<script> ::= { <element of script> }
<element of script> ::= <command> | <packet modification> | <variable
      modification> | <file inclusion> | <field definition>
```

Packet description

Script works with one packet which consists of *content* and *mask*. The content is a sequence of bytes, the mask is a set of conditions.

Packet's content is initially filled by 0. Packet mask has no conditions initially, i.e. corresponds to any packet.

You can modify the packet using several operators:

```
<packet modification> :: = <packet's content modification> | <packet mask
      modification>
<packet's content modification> ::= <field's name> ["="] <value>
```

Content modification means specifying some value for some field, ex:

```
srcport = 100
srcport 100
```

While content modification packet mask will also change: there will be added a new condition that the specified field must have the specified value. So the mask is based on content.

If you include some header file then packet's content is filled by correct values which are specified in header.

Actually in script you don't describe multiple packets but only one packet. Commands SEND and others can simply get the current packet's content and mask and then generate it or store in separate buffer. It looks like describing several packets but next packet inherits the content and mask of previous.

There is a byte pointer which is used in different operations. This pointer is initially equal to 0. After a <packet's content modification> it will be set just after the field and will be equal to (field's position + field's size).

Program also works with *packet's size*. This size may be used, for example, while packet generation. It will be equal to maximum position of byte pointer which was fixed from the start of script processing. So by default the size cannot decrease and next packet cannot be smaller than previous. One exception works: the size may decrease if the size of higher field decreases. The higher field is field with maximum position. It may have string type of value. Ex:

```
tcp.data "hello"
      SEND
tcp.data "hel"
      SEND                  // size of this packet will be smaller then size
```

of previous

The mask notion is described later.

Simple file inclusion

<file inclusion> ::= <name of file>

To include a file you may not use command INCLUDE but simply type the name of file. Program will search the specified file.

5.1. Types of values

Numbers

By default numbers are decimal, but while using option -h they will be hexadecimal.

If you want to explicitly specify the type of number then use 0x prefix for hexadecimal number and dot at the end for decimal numbers:

```
scrport = 0x8888     // hexadecimal number
ethproto = 5.        // decimal number
ethproto = 5         // decimal number if option –h is not used
```

The size of hexadecimal number will be determined by its form:

```
ttl = 0x8                  // one byte
ttl = 0x88           // one byte
ethproto = 0x188     // two bytes
```

For big numbers the number of digits must be even.

For decimal number you don't need to explicitly specify its size: the size will be determined by field's size.

```
srcport 4      // field 'srcport' has size = 2, so 4 means the value of two
      bytes
```

While field definition you must explicitly specify the size for decimal numbers:

```
.srcport      4s2    // after symbol 's' the size of number is specified
```

Strings

String is a set of characters enclosed with quotation marks or apostrophes.

The standart escape sequences are available:

```
\r, \n, \', \", \t, \a, \b, \x00.
```

If a string is enclosed with apostrophes then it's possible to insert references in this string. The reference is a name of field, variable, substitution (defined by DEFINE or GDEF command) or some special value which is enclosed with $.

```
'value of field = $name$'
```

The reference will be replaced by its value:

```
'value of field = 2'
```

Fields with string type of value don't have concrete size. So strings of any length may be specified as value. It may be changed by command SETSIZE.

IP the address

Examples:

```
.srcip 1.1.1.1
srcip localhost
srcip mycomp
srcip www.yandex.ru
```

While field definition must be used the first format: four numbers (indicates that the type of field is IP address). Further you can use DNS name.

MAC the address

Examples:

```
.srcmac 11:22:33:44:55:66
```

```
srcmac 11-22-33-44-55-66
srcmac 0x112233445566
srcmac 112233445566
```

While field definition must be used the first or second formats (indicates that the type of field is MAC address). Further you can use any format.

IPv6 address

```
ip6.src 1122:3344:5566:7788:99aa:bbcc:ddee:ff00
ip6.src 0:0:0:0:0:0cc:d00:0
```

DNS the name is not accepted.

5.2. Automatic values

There are some special fields for which values may be automaticly calculated.

Names of values: IPlen, IPcrc, IPv6len, TCPcrc, UDPcrc, UDPlen, ICMPcrc.

It is possible to specify such values for corresponding fields of network headers. Ex:

```
ip.crc = IPcrc
```

They are already specified in standart headers, so when generating, for example, TCP packet it will contain correctly calculated check sums.

While calculating some values, program will demand, that certain fields have been defined. For example, for values IPlen, IPcrc the field with name ip.ver must be defined. Its position points to beginning of IP header. The standard ip header (ip.fws) defines this field. Thus you can build own packets where ip header has not standart position. The position of redefined ip.ver field (see field definition) must point to actual position of ip header.

It is possible not to use automatic values. If you specify own value then it will be used:

```
ip.crc = 0
```

After some actions (command CLEAR, command GETPAC) all automatic values will be marked as no active so they will not be calculated. In this case after these actions you must again explicitly specify

```
ip.crc = IPcrc
ip.len = IPlen
```

5.3. Creation own headers

Sometimes it is necessary to define new fields for the description of headers of unknown network protocols, or simply to redefine the parameters of already known fields.

Field's parameters: position (as the displacement in bytes from the beginning of channel level header), size, type (i.e. the type of value).

<field definition> ::= "." <new field's name> <initial value>

The dot MUST NOT be separated by space from field's name.

Ex:

```
.new_field1 11                  // one-byte field, type: decimal number
.new_field2 0x1122          // two-byte field, type: hexadecimal number
.new_field3 11s2                // two-byte field, 11 in decimal format
.new_field4 'GET host'      // type: string
.new_field5 1.2.3.4         // type: IP address
```

The type of field and its size are defined by specified value. Packet's content will also change: the specified value will be written in packet's buffer.

The position of a newly created field is defined by the current position of byte pointer. After field definition byte pointer will be increased by the size of defined field. So it will point to position just after the defined field.

Command POS, PASS, BACK are used for manage byte pointer.

If some field with specified name already exists then its parameters will be rewritten corresponding the new definition.

Examples: headers/ip_header.fws, headers/tcp_header.fws.

5.4. Building packets

Full packet (packet's template) consists of several headers of different network protocols. Program enables you to build such complex templates using prepared headers.

Example:

```
include vlan
include ip_header
include tcp_header
```

In example above is shown how to build tcp packet with vlan header instead of simple Ethernet II header (which is defined in standart header "tcp.fws"). Parameters of fields from standart ip and tcp headers will be redefined.

5.5. Packet mask

Packet mask is used while comparison a packet from script with some packet from trace-file or packet received on physical interface. Mask is a set of conditions and is based on packet' content:

```
srcport = 40          // modifies content but also adds condition in mask
// that value of srcport must be equal to 40
```

While field definition mask is similarly changed.

Some special operations with mask are also available:

```
<packet mask modification> ::= <adding a new condition to mask> | <removing
      condition from mask>
<adding a new condition to mask> ::= <field's name> <comparison qualifier>
      <value>
     <comparison qualifier> ::= "!=" | ">" | "<" | ">=" | "<="
<removing condition from mask> ::= <field's name> ["="] "any"
```

Example:

```
srcport != 40         // adds condition that srcport must not be equal to 40
srcport > 20          // adds condition that srcport must be greater than 40
srcport = any         // removes from mask all the conditions related to field
      srcport
```

5.6. Variables

Variables are some values stored separately from packet's content. They may be changed by special commands and its names may be used in parameters to commands instead of concrete values.

Variables are created by command VAR. Ex:

```
VAR (v1, num, 1)
```

Each variable is associated with some field and the type of variable's value is the same as type of field's value. In the above example "num" is a field from TCP header (program knows this field a priory, you don't need to always include tcp header).

Created variables may modified by commands INCVAR, DECVAR, MULVAR, DIVVAR.

For modifying variables you can also use the following syntax:

```
<variable modification> ::= <variable's name> <operation with variable>
      <value>
   <operation with variable> ::= ["="] | "+=" | "-="
```

Ex:

```
v1 += 2        // adds 2 to variable
v1 -= 4        // subtract 2 from variable
v1 = 4  // sets value of variable
```

5.7. Commands and special values

<command> ::= <command's name> <command's parameters>

Parameters to command may be enclosed in round brackets.

Parameters to command may be separated by space or by commas.

Between the name of command and its parameters symbol = may be typed.

SEND
PARAMETERS: {accept | drop | any } In common regime generates the packet defined above. In other regimes (testing packet filter, see command FASTTEST and option -c) may simply separate packets one from another, so by this command the current content of buffer will be fixed and the new packet will be registered. The requests after command don't make sense in common regime (only while testing packet filter).
PAUSE
PARAMETERS: <number of milliseconds> Pauses the execution for a specified interval of time.
DROP
PARAMETERS: {accept | drop | any } Request specification. The request that the packet must not be recieved. May be used as command - replacement for "SEND DROP". It may be processed as command while testing packet filter only (command FASTTEST or option -c). In common regime it may be amoung parameters to command only.
ACCEPT
PARAMETERS: {accept | drop | any } Request specification. The request that the packet must be recieved. Analoge of SEND ACCEPT. It may be processed as command while testing packet filter only (command FASTTEST or option -c). In common regime it may be amoung parameters to command only.
ANY
PARAMETERS: {accept | drop | any } Request specification. No requests: the packet may be recieved or not. Analoge of SEND ANY. It may be processed as command while testing packet filter only (command FASTTEST or option -c). In common regime it may be amoung parameters to command only. This special word may also be used as value for field that means exlusion the all conditions with this field from current mask of packet - value of the field may be any.
REP

PARAMETERS: <the number of generation>

The next generation command (command SEND) will generate not one, but several packets.

RESET

PARAMETERS: no parameters

The mask of packet (the set of previously defined conditions) will be cleared. New mask will correspond to any packet. This command is usually contained in headers to make the mask correspond to all packets of given type (ex: TCP packets).

INC

PARAMETERS:

The field for which the value will be specified below becomes autoincremented. While generating several packets (command REP) the value of field will be incremented. See "samples/synflood.fws"

OFFSET

PARAMETERS: <number of bites>

The position of the next defined field will be shifted to the left for the given <number of bits> which must be from 1 to 7. So every written value will be shifter to the left before writing. Nevertheless, after the writing the left bits will be also changed and set to 0. To avoid this use command MASK. See "headers/tcp_header.fws"

DEFAULTS

PARAMETERS: {accept | drop | any | revers }

Defines default requests for packets. These requests will be applied when there are not enough explicitly defined requests for some packet (specified as parameters to command SEND, WAIT and its analoges). Initially default requests are ACCEPT ANY ANY... Command MI also changes default requests. See "samples/compare_mode1.fws".

INCLUDE

PARAMETERS: <name of file>

Starts processing the content of given file. The search of file will be performed in the current directory, all search paths (see option -I). For every path the content of samples, headers, traces folders will be also examined.

DEVICE

PARAMETERS: <type of device> {<name of interface>}

Reopens interfaces. The type of device: eth, ip, tcp. The name of device is the same as for -d option, depends on the type of device. New line terminates the list of names.

WAIT

PARAMETERS: {accept | drop | any }

Waits for packet whose mask is defined above. The command will finish work when such packet is recieved on waitable interface. The waitable interface is interface for which strict request (accept or drop) have been specified in parameters to command or in defaults (command DEFAULT). For TCP device the command will only wait data on the main interface. In the general case command may wait no one but several packets (added by OR command). If any of them is recieved then command terminates. Command waits packets until timeout expires (command TIMEOUT). See "samples/waiting_packets.fws".

EXIT

PARAMETERS: <status>

Terminates the execution. The status may be some decimal number. Value 0 is reserved for successfull test, 1 - fatal error, 2 - not successfull test, 3 - error while TCP connecting or listening (timeout expires).

PASS

PARAMETERS: <number of bytes>

Increases the byte pointer for the given <number of bytes>.

DEFINE

PARAMETERS: <name> <value>

Defines the substitution which will be applied while reading some values (in parameters to commands and others). <name> will be replaced by <value>. This substitution may be also performed in strings enclosed in apostrophes. In this case the <name> must be enclosed in $ (ex: 'value = $name$'. See also command GDEF.

MASK

PARAMETERS: <field's mask>

Defines the mask for the next defined field. Mask is hexadecimal number. Value for field will be written only in bits corresponding not null bits of mask. See "headers/tcp_header.fws".

POS

PARAMETERS: <new position> | <field's name>

Sets the <new position> of byte pointer. In the case of <field's name> new position will be equal to field's position.

BACK

PARAMETERS: <number of bytes>

Reduces the pointer for the given <number of bytes>.

CLEAR

PARAMETERS: no parameters

Clears info about the maximum size of previous packets. New packet may be smaller than previous ones. This command also makes all auto-calculated values inactive.

VAR

PARAMETERS: <name of variable> <name of field> <initial value> ("autoset"|["static"])

Command creates the new variable <name of variable> or reinitializes the old one if some variable of the same name is already exist. The newly created variable will have the same value's type as <name of field>. This command also sets the <initial value> for variable. Variable's value is stored separately from packet's buffer. The "autoset" type of variable indicates that the variable will be initialized by recieved packet (while using WAIT command or its analoges), i.e. from recieved packet will be obtained value of <name of field> and copied to variable. "static" type indicates that variable must not be changed while recieving packet. The "static" keyword may be ommited only if parameters to command are enclosed in round brackets.

The <name of variable> may appear amoung parameters to other commands. In this case it will be replaced by its value. Such a replacement will be also performed in strings enclosed in apostrophes. In this case the <name of variable> must be enclosed in $ (ex: 'value of variable = $name$').

See "samples/ask_mac.fws", "samples/variables.fws".

NAME

PARAMETERS: <name of packet>

Defines the name of currently described packet which will be displayed in report instead of not obvious "Packet on line ..."

MES

PARAMETERS: <string of message>

Defines the message which will be displayed the every time on recieving the currently described packet. Substitutions are allowed in the form of $name$. The 'name' may reference to the field's name, variable's name, someone defined by GDEF command. In the case of field's name field's value will be retrieved from the content of recieved packet.

EXTENDED

PARAMETERS: no parameters

Enables the extended regime. While specifying the requests for packet each request must be followed by the user number of interface.

MI

PARAMETERS: <the user number of interface>

Sets the main interface from which packets will be generated. Changes default requests (command DEFAULT): sets accept request for the new main interface and no requests for other interfaces.

REVERS

PARAMETERS: not command

Request specification. May only be given in parameters for DEFAULT command. Instructs to revers the request for every packet.

BEEP

PARAMETERS: no parameters

Plays the sound.

SAFETERM

PARAMETERS: no parameters

If the intensity of packets is very high and the program fails in deadlock on terminating, then this command will help. Terminating may become slower.

INTERVAL

PARAMETERS: <number of milliseconds>

Sets the value of interval between generating multiple packets while using command REP.

INCVAR

PARAMETERS: <name of variable> <value to add>

Increases the given <name of variable> for the specified <value to add>. The <value to add> may be negative.

OR

PARAMETERS: {accept | drop | any }

Analog of WAIT command. Adds the above packet to the set of packets which will be waited by command WAIT and its analoges. This command does not start waiting. Nevertheless, at once after adding packet may be registered as recieved. If some packet will be registered as recieved before call to WAIT (WAITALL) then command WAIT will terminate immediately.

GDEF

PARAMETERS: <new name> <original name>

Defines the substitution which will be applied while reading almost any read word from text. <New name> will be replaced by <original name>. This substitution may be also performed in strings

enclosed in apostrophes. In this case the name must be enclosed in $ (ex: 'value = $name$'.

TIMEOUT

PARAMETERS: <initval in milliseconds>

Defines the timeout for WAIT command (and its analoges), also for imitation of application's work. Null value means infinite timeout (such timeout will not be applied for imitation of application's work). In the case of negative value its absolute value will be obtained as timeout, but WAIT command (its analoges) will work differently: it will always wait for the whole timeout (not terminating on first recieved packet). So several packets may be registered as recieved. This command also defines the timeout for TCP server while waiting for connections.

QUIET

PARAMETERS: no parameters

Instructs to not display some annoying messages.

CYC

PARAMETERS: <number of iterations>

Command instructs that next WAIT command (its analoges) or next block of script will be processed by several times = <number of iterations>. The "inf" value is available which means infinite iterant processing.

SYSCALL

PARAMETERS: <command>

Implementes the system call. <command> must specify command's name (path to program) with parameters. Special value CALLRES may be used to obtain the status of last system call.

NOTDOUBLEMES

PARAMETERS: no parameters

Avoids displaying of double messages (specified by command MES). It also avoids the recieving of corresponding double packets, i.e. such packets will be ignored and don't cause WAIT command (its analoges) to terminate on them.

RAND

PARAMETERS: no command

Specifies the random value for field.

PRECISEWAIT

PARAMETERS: no parameters

After the work of WAIT command (its analoges) all trace threads will be blocked until the next call to WAIT command. So there will be no missed packets between subsequent calls to WAIT command.

TRACE

PARAMETERS: <name of trace file>

Opens the given trace file for subsequent work with it.

EP

PARAMETERS: <type> <user number of interface> <name of field> <field's value>

Defines an end point. While imitation of application's work the end point is a entity used for distinguishing between packets in trace file belonging to different sources (so they, for example, must be generated from different interfaces). All the packets for which the given <name of field> has the given <field's value> will belong to defined end point.

There are two <types> of end points: "recv" (recieving ep) and "gen" (generating ep). Generating end points search their packets in trace file and generate them. Recieving end points - wait for their packets. The packets from trace file are scaned in series. The generation can only be performed after recieving previous packets. The wait will be started after generation previous packets. The <user number of interface> specifies the interface from which packets will be generated or waited. See "headers/configSession"

RUN

PARAMETERS: <base request> <list of packets>

This command starts imitation of application's work. Parameters to command specify the request to result of test. <Base request> may be: drop, any, accept. List of packets, for example: 1,2,3-5,7-. Minus at the end of list means expanding to last packet in trace file. List "any" is equal to "1-".

The result for packets from list must correspond to base request. The result for packets not from list must correspond to inverted base request. Ex: "run accept 1-" means "all packets must be accepted", "run drop 6-" means "all packets before 6 must be accepted, rest of packets - droped", "run any any" means no requests. See "samples/convtest1"

WRITE

PARAMETERS: no parameters

Writes the trace file opened by TRACE command on disk

GETPAC

PARAMETERS: <number of packet>

Copies the specified packet from trace file to the buffer of current packet.

SETPAC

PARAMETERS: <number of packet>

Replaces the specified packet in trace file by the current packet.

INSPAC

PARAMETERS: <number of packet>

Insert the current packet in trace file, moving the all packets with given number and higher.

DELPAC

PARAMETERS: <number of packet>

Deletes the specified packet from trace file.

FULLMASK

PARAMETERS: no parameters

Fills the mask of packet so that the all fields will be included in mask. So while comparing packets the full packet's content will be compared. By default while describing packet's content the mask will be also added by new conditions, so the using of this command make sense only after the use of RESET command (this command is used in headers). It must be well realized that packets will be compared only by mask which is not always synchronized with packet's content.

IFR

PARAMETERS: <name of packet> "{" <block of script> "}"

Processes the block of script if the last recieved packet (command WAIT, its analoges) has the given name (which was specified by command NAME). "timeout" may be specified as the name of packet what means that the block must be processed in the case of timeout. Command will not distinguish newly added packets and old ones if they have the same name. Take a note of it when using UNFIX command. See also CLEARREG command.

PRINT

PARAMETERS: <message>

Displays the given message. Use symbole \n in message to indicate that line feed must be performed.

FILTER

PARAMETERS: <user number of interface> <filter string>

Sets the fast low-level filter (which is used by tcpdump) for the given interface. The format of filter is described in libpcap (WinPcap) or tcpdump documentation. See "samples/my_gateway"

UNFIX

PARAMETERS: no parameters

By default after the work of WAIT command (its analoges) the statuses for all waited packets will be fixed, so there may be no packets to wait for the next call to WAIT. This command marks these old packets as newly added. The previous status for them will be lost. Take a note of that ALL old

packets will be unfixed, so they will be waited: this may cause unexpected results. Consider the use of CLEARREG command.

WAITALL

PARAMETERS: no parameters

The analog of WAIT command. Doesn't add the previously defined packet to the list of waited ones. Starts waiting simply. Packets may be already added by OR command (or using of UNFIX command).

CHTRACE

PARAMETERS: "{" <block of script> "}"

The given block of script may contain field's values definitions or command PRINT. These definitions will be applied to every packet from trace file which corresponds the mask described before the command.

COPYREC

PARAMETERS: no parameters

The recieved packet (see command WAIT, its analoges) will be copied to the buffer of current packet. Precision waiting must be first enabled (command PRECISEWAIT).

FASTTEST

PARAMETERS: no parameters

Enables fasttest regime. See "samples/fasttest".

GETCH

PARAMETERS: no parameters

Waits for press <Enter>

NUMRET

PARAMETERS: <number of retransmitions>

While imitation of application's work if some packets have not been recieved for a long time (command TIMEOUT), then the previously generated packets will be retransmitted. One retransmission by default.

RM

PARAMETERS: <type> <name of field> <sought value> <value to set>

While imitation of application's work some values in packets from trace file may be automatically replaced before generating packet or before waiting one. So the <type> of replacement ("recv" or "gen") instructs when the replacement must be applied: before generating packet or forming the packet which will be waited.

So it is possible to generate one packet but wait another. It may be useful if packets are modified on their way.

The <name of field> specifies the field for which the replacement must be applied. The <sought value> is the value of field which will be sought in packets to replace it. It will be replaced by the given <value to set>. Some special values are allowed: "first" and "second". In this case the concrete value will be obtained from the first or second packet in trace file. See "headers/natConfigSession"

ARM

PARAMETERS: <name of field> <field's value>

Defines an adaptive replacement. This command gets two previously defined replacements (command RM), marks them as not active initially. While imitation of application's work program will wait for the first packet for which the given <name of field> has the given <field's value>. Then for each of two replacements program sets their <value to set>, copying it from the received packet, then marks replacements as active. So the test will be finally configured after receiving some packet only.

Note: from the received packet will be obtained value of that field which has been specified for the first replacement. Then this value will be copied to <value to set> of second replacement.

See "headers/natConfigSession"

RANGE

PARAMETERS: <number of start packet> <number of stop packet>

While imitation of application's work the work will be performed with packets (from trace file) which are within the given range. Null value for start packet means first packet in file. Null value for stop packet means last packet in file.

HELP

PARAMETERS: <name of command> | <part of the field's name>

Displays the description of the command. The "all" value is available to display info about all commands. Also displays the list of fields which have the given string in their name.

TIMED

PARAMETERS: no parameters

Imitation of application's work will be implemented with considering time stamps from trace file. The test may become slower.

DEFAULTTEST

PARAMETERS: no parameters

Sets the default parameters for imitation of application's work (timeout, number of retransmitions, packets range, timed mode), removes all previously added end points (command EP) replacements (command RM), adaptive replacements (command ARM), cieves (command CIEVE). In

short: full reset.

SETSIZE

PARAMETERS: <name of field> <decimal value of a new size of field>

Allows to specify the size for fields which don't have concrete size initially (strings). It can be also used to change the size for fields with concrete size (hexadecimal numbers). Value -1 may be used to specify the undefined size.

IF

PARAMETERS: <value1> <type of compare> <value2> "{" <first block of script> "}" ["else" "{" <second block of script> "}"]

Processes the first block of script if condition is met, otherwise processes the second block if it is specified. <Types of compare>: = (==), !=, >, <, >=, <=. Hexadecimals number are treated as strings (with 0x prefix). If you have problems try to watch how these values are represented by string using PRINT command for example.

CURPOS

PARAMETERS: no command

This special value allows to get the current value of byte pointer.

CURSIZE

PARAMETERS: no command

This special value allows to get the current size of packet.

GOTO

PARAMETERS: <value of any type> [<stop position>]

Performs the search of the given value in current packet. The search will be started from the current position of byte pointer. Value may has any type. The result of search is available through GOTORES keyword. In the case of successfull search the byte pointer will be moved to the found entry. Stop position may be equal to -1. It means search to the end of packet. Stop position may be ommited but in this case parameters must be enclosed in brackets.

GOTOB

PARAMETERS: <value of any type> [<stop position>]

Is similar to GOTO command but performs back search.

GOTORES

PARAMETERS: no command

This is a special value which allows to get the result of last search performed by GOTO or GOTOB command. 1 - successfull search, 0 - not successfull search.

SETPOS

PARAMETERS: <field's name> <decimal value of a new position>

Sets a new position for the given field.

DECVAR

PARAMETERS: <name of variable> <value to subtract>

Analoge of INCVAR. Subtracts the given value from variable.

BREAK

PARAMETERS: no parameters

Breaks the cycle caused by using CYC command.

RECV

PARAMETERS: no command

This special word specifies that some entity will perform its function upon recieving packet.

GEN

PARAMETERS: no command

This special word specifies that some entity will perform its function upon sending packet.

FIRST

PARAMETERS: no command

Retrieves field's value from first packet in trace file.

SECOND

PARAMETERS: no command

Retrieves field's value from second packet in trace file.

CIEVE

PARAMETERS: <name of field>

Causes that while imitation of application's work the value of specified field will not be considered when comparing waited packet with recieving one.

IFNDEF

PARAMETERS: <name of entity> "{" <script's block> "}"

Executes block if given entity has not been defined (entity: variable, field, someone defined by GDEF or DEFINE commands).

CALLRES

PARAMETERS: no command

This special value allows to get the result of last system call (command SYSCALL). Only for UNIX.

MULVAR

PARAMETERS: <name of variable> <multiplier>

Multiply given variable by specified value.

DIVVAR

PARAMETERS: <name of variable> <divisor>

Divide given variable by specified value.

SENDWAIT

PARAMETERS: <requests>

Analog of "OR <requests> SEND WAITALL". Purpose: the packet will be added to waited ones before its generation, so it will not be missed by sniffer.

NOTCOPYREC

PARAMETERS: no parameters

Reverses the action of COPYREC.

CLEARREG

PARAMETERS: no parameters

Clears info about all packets which were added (by WAIT, OR commands). They will not be displayed in final report. If this command is typed at the end of script then it avoids displaying final report.

SHOWREP

PARAMETERS: no parameters

Displays report.

LASTRES

PARAMETERS: no command

This special value enables to get the last result of analysing static performed by SHOWREP. 0 - successfull, 2 - some discrepancy.

CURTIME

PARAMETERS: no command

This special value allows to get current time.

PRINTL

PARAMETERS: <message>

Analog of PRINT command. Additionally performs line feed.

FIXMASK

PARAMETERS: no parameters

Fixes the mask of packet so that defining field's values (also fields definitions) doesn't cause its changing.

UNFIXMASK

PARAMETERS: no parameters

Performs action reversed to action of FIXMASK