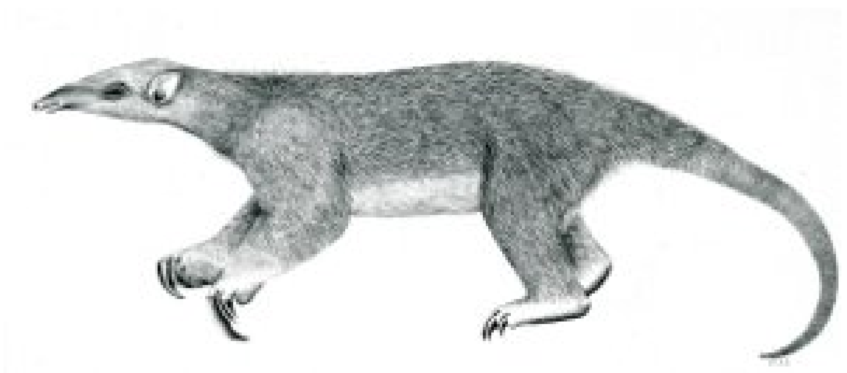
Tranalyzer2

Version 0.8.4, Beta, Tarantula

Flow based forensic and network troubleshooting traffic analyzer



Tranalyzer Development Team

Contents

1 Introduction

Tranalyzer2 is a lightweight flow generator and packet analyzer designed for simplicity, performance and scalability. The program is written in C and built upon the *libpcap* library. It provides functionality to pre- and post-process IPv4/IPv6 data into flows and enables a trained user to see anomalies and network defects even in very large datasets. It supports analysis with special bit coded fields and generates statistics from key parameters of IPv4/IPv6 Tcpdump traces either being live-captured from an Ethernet interface or one or several pcap files. The quantity of binary and text based output of Tranalyzer2 depends on enabled modules, herein denoted as **plugins**. Hence, users have the possibility to tailor the output according to their needs and developers can develop additional plugins independent of the functionality of other plugins.

1.1 Getting Tranalyzer

Tranalyzer can be downloaded from: `https://tranalyzer.com/downloads.html`

1.2 Dependencies

Tranalyzer2 requires **automake**, **libpcap** and **libtool**:

Kali/Ubuntu: `sudo apt-get install automake libpcap-dev libtool make zlib1g-dev`

Arch: `sudo pacman -S automake libpcap libtool zlib`

Fedora/Red Hat/CentOS: `sudo yum install automake libpcap libpcap-devel libtool zlib-devel bzip2`

Gentoo: `sudo emerge autoconf automake libpcap libtool zlib`

OpenSUSE: `sudo zypper install automake gcc libpcap-devel libtool zlib-devel`

Mac OS X: `brew install autoconf automake libpcap libtool zlib`[1]

1.3 Compilation

To build Tranalyzer2 and the plugins, run one of the following commands:

- Tranalyzer2 only:
  `cd "$T2HOME"; ./autogen.sh tranalyzer2`
  (alternative: `cd "$T2HOME/tranalyzer2"; ./autogen.sh`)

- A specific plugin only, e.g., `myPlugin`:
  `cd "$T2HOME"; ./autogen.sh myPlugin`
  (alternative 1: `cd "$T2PLHOME/myPlugin"; ./autogen.sh`)
  (alternative 2: `cd "$T2HOME/plugins/myPlugin"; ./autogen.sh`)

- Tranalyzer2 and a default set of plugins:
  `cd "$T2HOME"; ./autogen.sh`

[1]Brew is a packet manager for Mac OS X that can be found here: `https://brew.sh`

- Tranalyzer2 and all the plugins in `T2HOME`:
  `cd "$T2HOME"; ./autogen.sh -a`
- Tranalyzer2 and a custom set of plugins (listed in plugins.build) (Section 1.3.1):
  `cd "$T2HOME"; ./autogen.sh -b`

where `T2HOME` points to the root folder of Tranalyzer, i.e., where the file `README.md` is located.

For finer control of which plugins to load, refer to Section 2.2.

Note that if `t2_aliases` is installed, the `t2build` command can be used instead of `autogen.sh`. The command can be run from anywhere, so just replace the above commands with `t2build tranalyzer2`, `t2build myPlugin`, `t2build -a` and `t2build -b`. Run `t2build --help` for the full list of options accepted by the script.

1.3.1 Custom Build

The `-b` option of the `autogen.sh` script takes an optional file name as argument. If none is provided, then the default `plugins.build` is used. The format of the file is as follows:

- Empty lines and lines starting with a `'#'` are ignored (can be used to prevent a plugin from being built)
- One plugin name per row
- Example:

```
# Do not build the tcpStates plugin
#tcpStates

# Build the txtSink plugin
txtSink
```

A `plugins.ignore` file can also be used to prevent specific plugins from being built. A different filename can be used with the `-I` option.

1.4 Installation

The `-i` option of the `autogen.sh` script installs Tranalyzer in `/usr/local/bin` (as `tranalyzer`) and the man page in `/usr/local/man/man1`. Note that root rights are required for the installation.

Alternatively, use the file `t2_aliases` or add the following alias to your `~/.bash_aliases`:

```
alias tranalyzer="$T2HOME/tranalyzer2/src/tranalyzer"
```

where `T2HOME` points to the root folder of Tranalyzer, i.e., where the file `README.md` is located.

The man page can also be installed manually, by calling (as root):

```
mkdir -p /usr/local/man/man1 && gzip -c man/tranalyzer.1 > /usr/local/man/man1/tranalyzer.1.gz
```

1.4.1 Aliases

The file t2_aliases documented in $T2HOME/scripts/doc/scripts.pdf contains a set of aliases and functions to facilitate working with Tranalyzer. To install it, append the following code to ~/.bashrc or ~/.bash_aliases (make sure to replace $T2HOME with the actual path, e.g., $HOME/tranalyzer2-0.8.4):

```
if [ -f "$T2HOME/scripts/t2_aliases" ]; then
    . "$T2HOME/scripts/t2_aliases"            # Note the leading '.'
fi
```

1.5 Getting Started

Run Tranalyzer as follows:

```
tranalyzer -r file.pcap -w outfolder/outprefix
```

For a full list of options, use Tranalyzer -h or --help option: tranalyzer -h or tranalyzer --help or refer to the complete documentation.

1.6 Getting Help

1.6.1 Documentation

Tranalyzer and every plugin come with their own documentation, which can be found in the doc subfolder. The complete documentation of Tranalyzer2 and all the locally available plugins can be generated by running make in $T2HOME/doc. The file t2_aliases provides the function t2doc to allow easy access to the different parts of the documentation from anywhere.

1.6.2 Man Page

If the man page was installed (Section 1.4), then accessing the man page is as simple as calling

```
man tranalyzer
```

If it was not installed, then the man page can be invoked by calling

```
man $T2HOME/tranalyzer2/man/tranalyzer.1
```

1.6.3 Help

For a full list of options, use Tranalyzer -h option: tranalyzer -h

1.6.4 FAQ

Refer to the complete documentation in $T2HOME/doc for a list of frequently asked questions.

1.6.5 Contact

Any feedback, feature requests and questions are welcome and can be sent to the development team via email at:

tranalyzer@rdit.ch

2 Tranalyzer2

Tranalyzer2 is designed in a modular way. Thus, the packet flow aggregation and the flow statistics are separated. While the main program performs the header dissection and flow organisation, the plugins produce specialized output such as packet statistics, mathematical transformations, signal analysis and result file generation.

2.1 Supported Link-Layer Header Types

Tranalyzer handles most PCAP link-layer header types automatically. Some specific types can be analyzed by switching on flags in `linktypes.h`. The following table summarises the link-layer header types handled by Tranalyzer:

Linktype	**Description**	**Flags**
`DLT_C_HDLC`	Cisco PPP with HDLC framing	
`DLT_C_HDLC_WITH_DIR`	Cisco PPP with HDLC framing preceded by one byte direction	
`DLT_EN10MB`	IEEE 802.3 Ethernet (10Mb, 100Mb, 1000Mb and up)	
`DLT_FRELAY`	Frame Relay	
`DLT_FRELAY_WITH_DIR`	Frame Relay preceded by one byte direction	
`DLT_IEEE802_11`	IEEE802.11 wireless LAN	
`DLT_IEEE802_11_RADIO`	Radiotap link-layer information followed by an 802.11 header	
`DLT_IPV4`	Raw IPv4	
`DLT_IPV6`	Raw IPv6	
`DLT_JUNIPER_ATM1`	Juniper ATM1 PIC (experimental)	`LINKTYPE_JUNIPER=1`
`DLT_JUNIPER_ETHER`	Juniper Ethernet (experimental)	`LINKTYPE_JUNIPER=1`
`DLT_JUNIPER_PPPOE`	Juniper PPPoE PIC (experimental)	`LINKTYPE_JUNIPER=1`
`DLT_LINUX_SLL`	Linux “cooked” capture encapsulation	
`DLT_NULL`	BSD loopback encapsulation	
`DLT_PPI`	Per-Packet Information	
`DLT_PPP`	Point-to-Point Protocol	
`DLT_PPP_SERIAL`	PPP in HDLC-like framing	
`DLT_PPP_WITH_DIR`	PPP preceded by one byte direction	
`DLT_PRISM_HEADER`	Prism monitor mode information followed by an 802.11 header	
`DLT_RAW`	Raw IP	
`DLT_SYMANTEC_FIREWALL`	Symantec Enterprise Firewall	

2.2 Enabling/Disabling Plugins

The plugins are stored under *~/.tranalyzer/plugins*. This folder can be changed with the `-p` option.

By default, all the plugins found in the plugin folder are loaded. This behaviour can be changed by altering the value of `USE_PLLIST` in *loadPlugins.h:12*. The valid options are

`USE_PLLIST`	**Description**
0	load all plugins in the plugin folder (default)
1	use a whitelist (loading list)
2	use a blacklist

This following sections discuss the various ways to selectively enable/disable plugins.

2.2.1 Default

By default, all the files in the plugin folder named according to the following pattern are loaded:

```
^[0-9]{3}_[a-zA-Z0-9]+.so$
```

To disable a plugin, it must be removed from the plugin folder. A subfolder, e.g., *disabled*, can be used to store unused plugins.

2.2.2 Whitelisting Plugins

If `USE_PLLIST=1`, the whitelist (loading list) is searched under the plugins folder with the name `plugins.txt`. The name can be changed by adapting the value `PL_LIST` in *loadPlugins.h:13*. If the file is stored somewhere else, Tranalyzer2 `-b` option can be used.

The format of the whitelist is as follows (empty lines and lines starting with a `'#'` are ignored):

```
# This is a comment

# This plugin is whitelisted (will be loaded)
001_protoStats.so

# This plugin is NOT whitelisted (will NOT be loaded)
#010_basicFlow.so
```

Note that if a plugin is not present in the list, it will NOT be loaded.

2.2.3 Blacklisting Plugins

If `USE_PLLIST=2`, the blacklist is searched under the plugins folder with the name `plugins.txt`. The name can be changed by adapting the value `PL_LIST` in *loadPlugins.h:13*. If the file is stored somewhere else, Tranalyzer2 `-b` option can be used.

The format of the blacklist is as follows (empty lines and lines starting with a `'#'` are ignored):

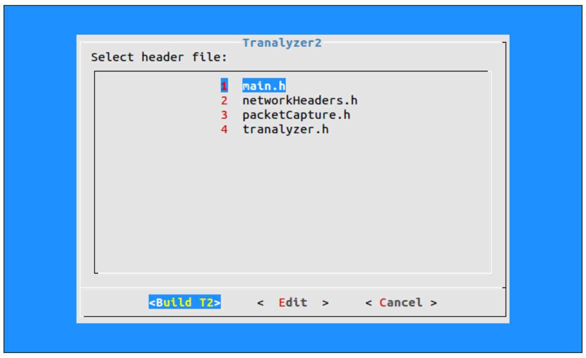
```
# This is a comment

# This plugin is blacklisted (will NOT be loaded)
001_protoStats.so

# This plugin is NOT blacklisted (will be loaded)
#010_basicFlow.so
```

2.2.4 Graphical Configuration and Building of T2 and Plugins

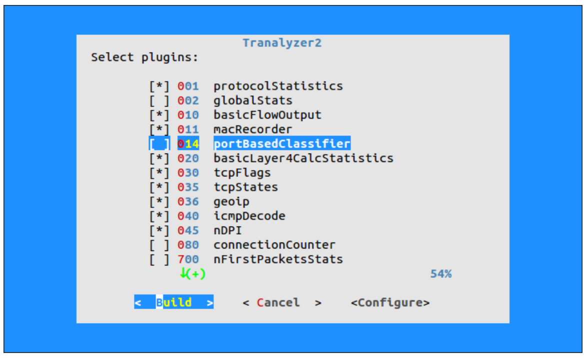
Tranalyzer2 comes with a script named `t2conf` allowing easy configuration of all the plugins through a command line based graphical menu:



```
Tranalyzer2
Select header file:

  1  main.h
  2  networkHeaders.h
  3  packetCapture.h
  4  tranalyzer.h

<Build T2>    <  Edit  >    < Cancel >
```



```
Tranalyzer2
Select plugins:

[*] 001  protocolStatistics
[ ] 002  globalStats
[*] 010  basicFlowOutput
[*] 011  macRecorder
[ ] 014  portBasedClassifier
[*] 020  basicLayer4CalcStatistics
[*] 030  tcpFlags
[*] 035  tcpStates
[*] 036  geoip
[*] 040  icmpDecode
[*] 045  nDPI
[ ] 080  connectionCounter
[ ] 700  nFirstPacketsStats
 ↓(+)                              54%

<  Build  >    < Cancel  >    <Configure>
```

Use the arrows on your keyboard to navigate up and down and between the buttons. The first window is only displayed if the `-t2` option is used. The `Edit` and `Configure` buttons will launch a text editor (`$EDITOR` or vim[2] if the environment variable is not defined). The second window can be used to activate and deactivate plugins (toggle the active/inactive state with the space key).

To access the script from anywhere, use the provided `install.sh` script, install `t2_aliases` or manually add the following alias to `~/.bash_aliases`:

```
alias t2conf="$T2HOME/scripts/t2conf/t2conf"
```

Where `$T2HOME` is the folder containing the source code of Tranalyzer2 and its plugins.

A man page for `t2conf` is also provided and can be installed with the `install.sh` script.

2.3 Man Page

If the man page was installed (Section 1.4), then accessing the man page is as simple as calling

```
man tranalyzer
```

If it was not installed, then the man page can be invoked by calling

```
man $T2HOME/tranalyzer2/man/tranalyzer.1
```

2.4 Invoking Tranalyzer

As stated earlier Tranalyzer2 either operates on Ethernet/DAG interfaces or pcap files. It may be invoked using a BPF if only certain flows are interesting. The required arguments are listed below. Note that the `-i`, `-r`, `-R` and `-D` options cannot be used at the same time.

2.4.1 Help

For a full list of options, use the `-h` option: `tranalyzer -h`

```
Tranalyzer 0.8.2 (Anteater), Tarantula - High performance flow based network traffic analyzer

Usage:
```

[2] The default editor can be changed by editing the variable `DEFAULT_EDITOR` (line 7)

```
    tranalyzer [OPTION...] <INPUT>

Input:
    -i IFACE     Listen on interface IFACE
    -r PCAP      Read packets from PCAP file or from stdin if PCAP is "-"
    -R FILE      Process every PCAP file listed in FILE
    -D EXPR[:SCHR][,STOP]
                 Process every PCAP file whose name matches EXPR, up to an
                 optional last index STOP. If STOP is omitted, then Tranalyzer
                 never stops. EXPR can be a filename, e.g., file.pcap0, or an
                 expression, such as "dump*.pcap00", where the star matches
                 anything (note the quotes to prevent the shell from
                 interpreting the expression). SCHR can be used to specify the
                 the last character before the index (default: 'p')

Output:
    -w PREFIX    Append PREFIX to any output file produced. If omitted, then
                 output is diverted to stdout
    -W PREFIX[:SIZE][,START]
                 Like -w, but fragment flow files according to SIZE, producing
                 files starting with index START. SIZE can be specified in bytes
                 (default), KB ('K'), MB ('M') or GB ('G'). Scientific notation,
                 i.e., 1e5 or 1E5 (=100000), can be used as well. If a 'f' is
                 appended, e.g., 10Kf, then SIZE denotes the number of flows.
    -l           Print end report in PREFIX_log.txt instead of stdout
    -s           Packet forensics mode

Optional arguments:
    -p PATH      Load plugins from path PATH instead of ~/.tranalyzer/plugins
    -b FILE      Use plugin list FILE instead of plugin_folder/plugins.txt
    -e FILE      Creates a PCAP file by extracting all packets belonging to
                 flow indexes listed in FILE
    -f FACTOR    Sets hash multiplication factor
    -x ID        Sensor ID
    -c CPU       Bind tranalyzer to one core. If CPU is 0 then OS selects the
                 core to bind
    -F FILE      Read BPF filter from FILE

    -v           Show the version of the program and exit

    -h           Show help options and exit

Remaining arguments:
    BPF          Berkeley Packet Filter command, as in tcpdump
```

2.4.2 –i INTERFACE

Capture data from an Ethernet interface `INTERFACE` (requires *root* privileges). If high volume of traffic is expected, then enable internal buffering in ioBuffer.h.

```
tranalyzer -i eth0 -w out
```

2.4.3 –r FILE

Capture data from a pcap file `FILE`.

```
tranalyzer -r file.pcap -w out
```

The special file `'-'` can be used to read data from *stdin*. This can be used, e.g., to process compressed pcap files, e.g., *file.pcap.gz*, using the following command:

```
zcat file.pcap.gz | tranalyzer -r - -w out
```

2.4.4 –R FILE

Process all the pcap files listed in `FILE`. All files are being treated as one large file. The life time of a flow can extend over many files. The processing order is defined by the location of the filenames in the text file. The absolute path has to be specified. The `gpl` script documented in `$T2HOME/scripts/scripts.pdf` can be used to generate such a list. All lines starting with a `'#'` are considered as comments and thus ignored.

```
cd ~/pcap/
$T2HOME/scripts/gpl > pcap_list.txt
tranalyzer -R pcap_list.txt -w out
```

2.4.5 –D FILE[*][.ext]#1[:SCHR][,#2]

Process files in a directory using file start and stop index, defined by `#1` and `#2` respectively. `ext` can be anything, e.g., `.pcap`, and can be omitted. If `#2` is omitted and not in round robin mode, then Tranalyzer2 never stops and waits until the next file in the increment is available. If leading zeroes are used, `#2` defaults to $10^{\text{number\_length}} - 1$. Note that only the last occurence of `SCHR` is considered, e.g., if `SCHR='p'`, then out.pca**p**001 will work, but out001pca**p**, will not. with the `:[SCHR]` option a new separation character can be set, superseeding `SCHR` defined in tranalyzer.h.

The following variables in tranalyzer.h can be used to configure this mode:

Name	Default	Description
`RROP`	0	Activate round robin operations WARNING: if set to 1, then findexer will not work anymore
`POLLTM`	5	Poll timing (in seconds) for files
`MFPTMOUT`	0	> 0: timeout for poll > POLLTM, 0: no poll timout
`SCHR`	`'p'`	Separating character for file number

For example, when using `tcpdump` to capture traffic from an interface (eth0) and produce 100MB files as follows:

```
sudo tcpdump -C 100 -i eth0 -w out.pcap
```

The following files are generated: *out.pcap, out.pcap1, out.pcap2, . . . , out.pcap10, . . .*

Then `SCHR` must be set to `'p'`, i.e., the last character before the file number (out.pca**p**NUM) and Tranalyzer must be run as follows:

```
tranalyzer -D out.pcap -w out
```

Or to process files 10 to 100:

```
tranalyzer -D out.pcap10,100 -w out
```

Or to process files 10 to 100 in another format:

```
tranalyzer -D out10.pcap,100 -w out
```

Or to process files from 0 to $2^{32}-1$ using regex characters:

```
tranalyzer -D "out*.pcap" -w out
```

The last command can be shortened further, the only requirement being the presence of `SCHR` (the last character before the file number) in the pattern:

```
tranalyzer -D "*p" -w out
```

Note the quotes (`"`) which are necessary to avoid preemptive interpretation of regex characters and `SCHR` which MUST appear in the pattern. The same configuration can be used for filenames using one or more leading zeros, e.g., *out.pcap000, out.pcap001, out.pcap002, . . . , out.pcap010, . . .*

The following table summarises the supported naming patterns and the configuration required:

Filenames	**SCHR**	**Command**
out.pca**p**, out.pcap**1**, out.pcap**2**, . . .	`'p'`	`tranalyzer -D out.pcap -w out`
out.pca**p**00, out.pca**p**01, out.pca**p**02, . . .	`'p'`	`tranalyzer -D out.pcap00 -w out`
ou**t**0.pcap, ou**t**1.pcap, ou**t**2.pcap, . . .	`'t'`	`tranalyzer -D out0.pcap -w out`
ou**t**00.pcap, ou**t**01.pcap, ou**t**02.pcap, . . .	`'t'`	`tranalyzer -D out00.pcap -w out`
out_24.04.2016.20h00.pca**p**, out_24.04.2016.20h00.pca**p**1, . . .	`'p'`	`tranalyzer -D "out*.pcap" -w out`
out_24.04.2016.20h00.pca**p**00, out_24.04.2016.20h00.pca**p**01, . . .	`'p'`	`tranalyzer -D "out*.pcap00" -w out`
ou**t**0.pcap, ou**t**1.pcap, ou**t**2.pcap, . . .	`'t'`	`tranalyzer -D out0.pcap:t -w out`
out.pca**p**00, out.pca**p**01, out.pca**p**02, . . .	`'p'`	`tranalyzer -D out.pcap00:p -w out`

2.4.6 –w PREFIX

Use a `PREFIX` for all output file types. The number of files being produced vary with the number of activated plugins. The file suffixes are defined in the file tranalyzer.h (see Section 2.9.13) or in the header files for the plugins. If you forget to specify an output file, Tranalyzer will use the input interface name or the file name as file prefix and print the flows to *stdout*. Thus, Tranalyzer output can be piped into other command line tools, such as netcat in order to produce centralized logging to another host or an AWK script for further post processing without intermediate writing to a slow disk storage.

2.4.7 –W PREFIX[:SIZE][,START]

This option allows the fragmentation of flow files produced by Tranalyzer independent of the input mode. The expression before the `':'` is the output prefix, the expression after the `':'` denotes the maximal file size for each fragment and the number after the `','` denotes the start index of the first file. If omitted it defaults to `0`. The size of the files can be specified in bytes (default), KB (`'K'`), MB (`'M'`) or GB (`'G'`). Scientific notation, i.e., 1e5 or 1E5 (=100000), can be used as well. If no size is specified, the default value of 500MB, defined by `OFRWFILELN` in tranalyzer.h is used. If no size is specified, then the `':'` can be omitted. The same happens if no start index is specified. If an additional `'f'` is appended the unit is flow count. This enables the user to produce file chunks containing the same amount of flows. Some typical examples are shown below.

Command	Fragment Size	Start Index	Output Files
`tranalyzer -r nudel.pcap -W out:1.5E9,10`	1.5GB	10	out10, out11, …
`tranalyzer -r nudel.pcap -W out:1.5e9,5`	1.5GB	5	out5, out6, …
`tranalyzer -r nudel.pcap -W out:1.5G,1`	1.5GB	1	out1, out2, …
`tranalyzer -r nudel.pcap -W out:5000K`	0.5MB	0	out0, out1, …
`tranalyzer -r nudel.pcap -W out:5Kf`	5000 Flows	0	out0, out1, …
`tranalyzer -r nudel.pcap -W out:180M`	180MB	0	out0, out1, …
`tranalyzer -r nudel.pcap -W out:2.5G`	2.5GB	0	out0, out1, …
`tranalyzer -r nudel.pcap -W out,5`	`OFRWFILELN`	0	out0, out1, …
`tranalyzer -r nudel.pcap -W out`	`OFRWFILELN`	0	out0, out1, …

2.4.8 –l

All Tranalyzer command line and report output is diverted to the log file: `PREFIX_log.txt`. Fatal error messages still appear on the command line.

2.4.9 –s

Initiates the packet mode, where a file with the suffix `PREFIX_packets.txt` is created. The content of the file depends on the plugins loaded. The display of the packet number (first column is controlled by `SPKTMD_PKTNO` in main.h. The layer 7 payload can be displayed in hexadecimal and/or as characters by using the `SPKTMD_PCNTH` and `SPKTMD_PCNTC` respectively. A tab separated header description line is printed at the beginning of the packet file. The first two lines then read as follows:

```
%pktNo    time    pktIAT    duration    flowInd    flowStat    numHdrDesc    hdrDesc    ethVlanID
       ethType    srcMac    dstMac    srcIP4    srcPort    dstIP4    dstPort    l4Proto    ipTOS
       ipID    ipIDDiff    ipFrag    ipTTL    ipHdrChkSum    ipCalChkSum    l4HdrChkSum
    l4CalChkSum    ipFlags    pktLen    ipOptLen    ipOpts    seq    ack    seqDiff    ackDiff
    seqPktLen    ackPktLen    tcpStat    tcpFlags    tcpAnomaly    tcpWin    tcpOptLen    tcpOpts
       l7Content
...
25    1291753225.446846    0.000000    0.000000    23    0x00006000    6    eth:vlan:mpls{2}:ipv4:
    tcp    20    0x0800    00:90:1a:41:fa:45    00:13:c4:52:4a:07    188.62.56.56    62701
    212.243.221.241    80    6    0x00    0x26f6    0    0x4000    62    0x6ca6    0x6ca6    0
    x0247    0x0247    0x0040    460    0        0xb2a08909    0x90314073    0    0    0    0    0
    x59    0x18    0x0000    65535    12    0x01 0x01 0x08 0x0a 0x29 0x2d 0xc3 0x6e 0x83 0x63 0xc5
     0x76    GET /images/I/01TnJ0+mhnL.png HTTP/1.1\r\nHost: ecx.images-amazon.com\r\nUser-Agent:
    Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; de; rv:1.9.2.8) Gecko/20100722 Firefox/3.6.8\r
    \nAccept: image/png,image/*;q=0.8,*/*;q=0.5\r\nAccept-Language: de-de,de;q=0.8,en-us;q=0.5,en;
    q=0.3\r\nAccept-Encoding: gzip,deflate\r\nAccept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\
    nKeep-Alive: 115\r\nConnection: keep-alive\r\nReferer: http://z-ecx.images-amazon.com/images/I
    /11J5cf408UL.css\r\n\r\n
...
```

2.4.10 –p FOLDER

Changes the plugin folder from standard *~/.tranalyzer/plugins* to `FOLDER`.

2.4.11 –b FILE

Changes the plugin blacklist file from *plugin_folder/plugin_blacklist.txt* to `FILE`, where `plugin_folder` is either *~/.tranalyzer/plugins* or the folder specified with the `-p` option.

2.4.12 –e FLOWINDEXFILE

Denotes the filename and path of the flow index file when the `pcapd` plugin is loaded. The path and name of the pcap file depends on `FLOWINDEXFILE`. If omitted the default names for the PCAP file are defined in *pcapd.h*. The format of the `FLOWINDEXFILE` is a list of 64 bit flow indices which define the packets to be extracted from the pcap being read by the `-r` option. In general the user should use a plain file with the format displayed below:

```
# Comments (ignored)
% Flow file info (ignored)
30
3467
656697
5596
```

For more information on the `pcapd` plugin please refer to its documentation.

2.4.13 –f HASHFACTOR

Sets and superseeds the `HASHFACTOR` constant in tranalyzer.h.

2.4.14 –x SENSORID

Each T2 can have a separate sensor ID which can be listed in a flow file in order to differentiate flows originating from several interfaces during post processing, e.g., in a DB. If not specified `T2_SENSORID` (`666`), defined in tranalyzer.h, will be the default value.

2.4.15 –c CPU

Bind Tranalyzer to core number `CPU`; if `CPU == 0` then the operating system selects the core to bind.

2.4.16 –F FILE

Read BPF filter from FILE. A filter can span multiple lines and can be commented using the `'#'` character (everything following a `'#'` is ignored).

2.4.17 BPF Filter

A Berkeley Packet Filter (BPF) can be specified at any time in order to reduce the amount of flows being produced and to increase speed during life capture ops. All rules of pcap BPF apply.

2.5 ioBuffer.h

Name	Default	Description
`ENABLE_IO_BUFFERING`	0	Enables buffering of the packets in a queue
If `ENABLE_IO_BUFFERING == 1`, the following flags are available:		
`IO_BUFFER_FULL_WAIT_MS`	200	Number of milliseconds to wait if queue is full
`IO_BUFFER_SIZE`	8192	Maximum number of packets that can be stored in the buffer (power of 2)
`IO_BUFFER_MAX_MTU`	2048	Maximum size of a packet (divisible by 4)

2.6 main.h

The monitoring mode can be configured with the following constants:

Name	Default	Description
`SPKTMD_PKTNO`	1	Print the packet number
`SPKTMD_PCNTC`	1	Print L7 content as characters
`SPKTMD_PCNTH`	0	Print L7 content as hex
`MIN_MAX_ESTIMATE`	0	Min/Max bandwidth statistics
The following flags control the monitoring mode:		
`MONINTTHRD`	1	Activate threaded interrupt handling
`MONINTBLK`	0	Block interrupts in main loop during packet processing, disables `MONINTTHRD`
`MONINTSYNC`	1	Synchronized print statistics
`MONINTTMPCP`	0	0: real time base, 1: pcap time base
`MONINTTMPCP_ON`	0	Startup monitoring. 1: on 0: off (require `MONINTTMPCP=1`)
`MONINTV`	1	≥ 1 sec interval of monitoring output if `USR2` is sent or `MONINTTMPCP=0`
`MONPROTMD`	1	0: report protocol numbers; 1: report protocol names
`MONPROTFL`	`"proto.txt"`	proto file

The `MONPROTL2` and `MONPROTL3` flags can be used to configure the L2 and L3 protocols to monitor. Their default values are

- `MONPROTL2`: `0x0042,0x00fe,ETHERTYPE_ARP,ETHERTYPE_RARP,ETHERTYPE_IP,ETHERTYPE_IPV6`
- `MONPROTL3`: `L3_ICMP,L3_IGMP,L3_TCP,L3_UDP,L3_GRE,L3_ICMP6,L3_SCTP`

2.7 networkHeaders.h

Name	Default	Description	Flags
`IPV6_ACTIVATE`	2	0: IPv4 only 1: IPv6 only 2: Dual mode	
`ETH_ACTIVATE`	1	0: No Ethernet flows 1: Activate Ethernet flows 2: Also use Ethernet addresses for IPv4/6 flows	
`SCTP_ACTIVATE`	0	SCTP protocol decoder for stream $\rightarrow$ flow generation is activated	
`SCTP_STATFINDEX`	0	1: findex constant for all SCTP streams in a packet 0: findex increments	`SCTP_ACTIVATE=1`
`MULTIPKTSUP`	0	Multi-packet suppression (discard duplicated packets)	`IPV6_ACTIVATE=0`
`T2_PRI_HDRDESC`	1	1: keep track of the headers traversed	
`T2_HDRDESC_AGGR`	1	1: aggregate repetitive headers, e.g., `vlan{2}`	`T2_PRI_HDRDESC=1`
`T2_HDRDESC_LEN`	128	max length of the headers description	`T2_PRI_HDRDESC=1`

2.8 packetCapture.h

The config file *packetCapture.h* provides control about the packet capture and packet structure process of Tranalyzer2. The most important fields are described below. Please note that after changing any value in define statements a rebuild is required. Note that the PACKETLENGTH switch controles the packetLength variable in the packet structure, from where the packet length is measured from. So statistical plugins such as basicStats can have a layer dependent output. If only L7 length is needed, use the packetL7length variable in the packet structure.

Name	Default	Description	Flags
`PACKETLENGTH`	3	0: including L2, L3 and L4 header 1: including L3 and L4 header 2: including L4 header 3: only higher layer payload (Layer 7)	
`FRGIPPKTLENVIEW`	1	0: IP header stays with 2nd++ fragmented packets 1: IP header stripped from 2nd++ fragmented packets	`PACKETLENGTH=1`
`NOLAYER2`	0	0: Automatic L3 header discovery 1: Manual L3 header positioning	
`NOL2_L3HDROFFSET`	0	Offset of L3 header	`NOLAYER2=1`
`MAXHDRCNT`	5	Maximal header count (MUST be ≥ 3)	`IPV6_ACTIVATE=1`

2.9 tranalyzer.h

Name	Default	Description	Flags
`REPSUP`	0	Activate alive mode	
`PID_FNM_ACT`	0	Save the PID into a file `PID_FNM` (default: `"tranalyzer.pid"`)	
`DEBUG`	0	0: no debug output 1: debug output which occurs only once or very seldom 2: + debug output which occurs in special situations, but not regularly 3: + debug output which occurs regularly (every packet)	
`VERBOSE`	2	0: no output 1: basic pcap report 2: + full traffic statistics 3: + info about fragmentation anomalies	
`MEMORY_DEBUG`	0	0: no memory debug 1: detect leaks and overflows (see *utils/memdebug.h*)	
`NO_PKTS_DELAY_US`	1000	If no packets are available, sleep for *n* microseconds	
`NON_BLOCKING_MODE`	1	Non-blocking mode	
`MAIN_OUTPUT_BUFFER_SIZE`	1000000	Size of the main output buffer	
`SNAPLEN`	`BUFSIZ`	Snapshot length (live capture)	
`CAPTURE_TIMEOUT`	1000	Read timeout in milliseconds (live capture)	
`ENABLE_BPF_OPTIMIZATION`	0	Optimize BPF filters	
`TSTAMP_PREC`	0	Timestamp precision: 0: microseconds, 1: nanoseconds	
`TSTAMP_UTC`	1	Time representation: 0: localtime, 1: UTC	
`TSTAMP_R_UTC`	0	Time report representation: 0: localtime, 1: UTC	
`ALARM_MODE`	0	Only output flow if an alarm-based plugin fires	
`ALARM_AND`	0	0: logical OR, 1: logical AND	`ALARM_MODE=1`
`FORCE_MODE`	0	Parameter induced flow termination (NetFlow mode)	
`BLOCK_BUF`	0	Block unnecessary buffer output when non Tranalyzer format event based plugins are active	

Name	Default	Description	Flags
PLUGIN_REPORT	1	Enable plugins to contribute to Tranalyzer end report	
DIFF_REPORT	0	0: absolute Tranalyzer command line USR1 report 1: differential report	
MACHINE_REPORT	0	USR1 report: 0: human compliant, 1: machine compliant	
REPORT_HIST	0	Store statistical report history in REPORT_HIST_FILE after shutdown and reload it when restarted	
ESOM_DEP	0	Allow plugins to globally access plugin dependent variables	
AYIYA	1	Activate AYIYA processing	
GENEVE	1	Activate GENEVE processing	
TEREDO	1	Activate TEREDO processing	
L2TP	1	Activate L2TP processing	
GRE	1	Activate GRE processing	
GTP	1	Activate GTP (GPRS Tunneling Protocol) processing	
VXLAN	1	Activate VXLAN (Virtual eXtensible Local Area Network) processing	
IPIP	1	Activate IPv4/6 in IPv4/6 processing	
ETHIP	1	Activate Ethernet within IP IPv4/6 processing	
CAPWAP	1	Activate CAPWAP processing	
CAPWAP_SWAP_FC	1	Swap frame control (required for Cisco)	
LWAPP	1	Activate LWAPP processing	
LWAPP_SWAP_FC	1	Activate LWAPP processing (required for Cisco)	
FRAGMENTATION	1	Activate fragmentation processing	
FRAG_HLST_CRFT	1	Enables crafted packet processing	FRAGMENTATION=1
FRAG_ERROR_DUMP	0	Dumps flawed fragmented packet to stdout	FRAGMENTATION=1
IPVX_INTERPRET	0	Interpret bogus IPvX packets	
ETH_STAT_MODE	0	Whether to use the innermost (0) or outermost (1) layer 2 type for the statistics	
RELTIME	0	0: absolute time 1: relative time	
FDURLIMIT	0	If > 0, force flow life span to $n \pm 1$ seconds	
FLOW_TIMEOUT	182	Flow timeout after a packet is not seen after n seconds	
HASH_AUTOPILOT	1	0: terminate when main hash map is full 1: flushes oldest NUMFLWRM flow(s) when main hash is full	
NUMFLWRM	1	Number of flows to flush when main hash map is full	HASH_AUTOPILOT=1

2.9.1 -D constants

the following constants influence the file name convention:

Name	Default	Description
RROP	0	round robin operations
POLLTM	5	poll timing for files
SCHR	'p'	separating character for file number

2.9.2 alive signal

The alive signal is a derivate of the passive monitoring mode by the USR1 signal, where the report is deactivated. If `REPSUP=1` then only the command defined by `REPCMDAS/W` is sent to the control program defined by `ALVPROG` as defined below:

Name	Default	Description
`REPSUP`	0	0: alive mode off, 1: alive mode on, monitoring report suppressed
`ALVPROG`	`"t2alive"`	name of control program

`REPCMDAS`	`"a=‘pgrep ALVPROG‘; \`
	`if [ $a ]; then kill -USR1 $a; fi"`
		alive and stall USR1 signal (no packets)

`REPCMDAW`	`"a=‘pgrep ALVPROG‘; \`
	`if [ $a ]; then kill -USR2 $a; fi"`
		alive and well USR2 signal (working)

If T2 crashes or is stopped a syslog message is issued by the t2alive deamon. Same if T2 gets started.

2.9.3 FORCE_MODE

A 1 enables the force mode which enables any plugin to force the output of flows independent of the timeout value. Hence, Cisco NetFlow similar periodic output can be produced or overflows of counters can produce a flow and restart a new one.

2.9.4 ALARM_MODE

A 1 enables the alarm mode which differs from the default flow mode by the plugin based control of the Tranalyzer core flow output. It is useful for classification plugins generating alarms, thus emulating alarm based SW such as Snort, etc. The default value is 0. The plugin sets the global output suppress variable `supOut=1` in the *onFlowTerminate()* function before any output is generated. This mode also allows multiple classification plugins producing an ‘AND’ or an ‘OR’ operation if many alarm generating plugins are loaded. The variable `ALARM_AND` controls the logical alarm operation. A sample code which has to be present at the beginning of the *onFlowTerminate()* function is shown below:

```
#if ALARM_MODE == 1
#if ALARM_AND == 0
        if (!Alarm) supOut = 0;
#else // ALARM_AND == 1
        if (!Alarm) {
                supOut = 1;
                return;
        }
#endif // ALARM_AND
#endif // ALARM_MODE == 1
```

Figure 1: *A sample code in the* onFlowTerminate() *routine*

2.9.5 BLOCK_BUF

if set to ‘1’ unnecessary buffered output from all plugins is blocked when non Tranalyzer format event based plugins are active: e.g. syslog, arcsight and text-based or binary output plugins are not loaded.

2.9.6 Report Modes

Tranalyzer provides a user interrupt based report and a final report. The interrupt based mode can be configured in a variety of ways being defined below.

Name	Default	Description
`PLUGIN_REPORT`	0	enable plugins to contribute to the tranalyzer command line end report
`DIFF_REPORT`	0	1: differential, 0: Absolute tranalyzer command line `USR1` report
`MACHINE_REPORT`	0	`USR1` Report 1: machine compliant; 0: human compliant

The following interrupts are being caught by Tranalyzer2:

Signal Name	Description
`SIGINT`	like `^C` terminates new flow production[3]
`SIGTERM`	terminates tranalyzer
`SIGUSR1`	prints statistics report
`SIGUSR2`	toggles repetitive statistics report

2.9.7 State and statistical save mode

T2 is capable to preserve its internal statistical state and certain viable global variables, such as the findex.

Name	Default	Description
`REPORT_HIST`	0	Store statistical report history after shutdown, reload it upon restart
`REPORT_HIST_FILE`	`"stat_hist.txt"`	default statistical report history filename

The history file is stored by default under `./tranalyzer/plugins` or under the directory defined by a `-p` option.

2.9.8 L2TP

A ‘1’ activates the L2TP processing of the Tranalyzer2 core. All L2TP headers either encapsulated in MPLS or not will be processed and followed down via PPP headers to the IP header and then passed to the IP processing. The default value of the variable is ‘0’. Then the stack will be parsed until the first IP header is detected. So all L2TP UDP headers having src and dest port 1701 will be processed as normal UDP packets.

2.9.9 GRE

A ‘1’ activates the L3 General Routing Encapsulation (L4proto=47) processing of the Tranalyzer2 core. All GRE headers either encapsulated in MPLS or not will be processed and followed down via PPP headers to the IP header and then passed to the IP processing. The default value of the variable is 0. Then the stack will be parsed until the first IP header is detected. If the following content is not existing or compressed the flow will contain only L4proto = 47 information.

[3] If two `SIGINT` interrupts are being sent in short order Tranalyzer will be terminated instantly.

2.9.10 FRAGMENTATION

A '1' activates the fragmentation processing of the Tranalyzer2 core. All packets following the header packet will be assembled in the same flow. The core and the plugin tcpFlags will provide special flags for fragmentation anomalies. If `FRAGMENTATION` is set to 0 only the initial fragment will be processed; all later fragments will be ignored.

2.9.11 FRAG_HLST_CRFT

A '1' enables crafted packet processing even when the lead fragment is missing or packets contain senseless flags as being used in attacks or equipment failure.

2.9.12 FRAG_ERROR_DUMP

A '1' activates the dump of packet information on the command line for time based identification of ill-fated or crafted fragments in tcpdump or wireshark. It provides the Unix timestamp, the six tuple, IPID and fragID as outlined in figure below.

```
MsgType   msg   time   vlan   srcIP   srcPort   dstIP   dstPort   proto   fragID   fragOffset
[WRN] packetCapture: 1. frag not found @ 1291753225.449690 20 92.104.181.154 42968 93.144.66.3
    52027 17 - 0x191F 0x00A0
[WRN] packetCapture: 1. frag not found @ 1291753225.482611 20 92.104.181.154 43044 93.144.66.3
    1719 17 - 0x1922 0x00A0
[WRN] packetCapture: 1. frag not found @ 1291753225.492830 20 92.104.181.154 55841 93.144.66.3
    28463 17 - 0x1923 0x00A0
[WRN] packetCapture: 1. frag not found @ 1291753225.503955 20 92.104.181.154 25668 93.144.66.3
    8137 17 - 0x1924 0x00A0
[WRN] packetCapture: 1. frag not found @ 1291753225.551094 20 92.105.93.227 41494 24.218.128.232
    27796 17 - 0x5A21 0x00A0
[WRN] packetCapture: 1. frag not found @ 1291753225.639627 20 86.51.18.243 38824 92.105.108.208
    55133 17 - 0x0DAE 0x00AC
```

Figure 2: *A sample report on stdout for packets with an elusive first fragment*

WARNING: If `FRAG_HLST_CRFT == 1` then every fragmented headerless packet will be reported!

2.9.13 *_SUFFIX

This constant defines the suffix of all plugin output files. For example if you specify the output *foo.foo* (with the `-w` option), the generated file for the per-packet output will be in the default setting *foo.foo_packets*.

2.9.14 RELTIME

`RELTIME` renders all time based plugin output into relative to the beginning of the pcap or start of packet capture. In `-D` or `-R` read operation the first file defines the start time.

2.9.15 FLOW_TIMEOUT

This constant specifies the default time in seconds (182) after which a flow will be considered as terminated since the last packet is captured. Note: Plugins are able to change the timeout values of a flow. For example the `tcpStates` plugin adjusts the timeout of a flow according to the TCP state machine. A reduction of the flow timeout has an effect on the necessary flow memory defined in `HASHCHAINTABLE_SIZE`, see below.

2.9.16 FDURLIMIT

`FDURLIMIT` defines the maximum flow duration in seconds which is then forced to be released. It is a special force mode for the duration of flows and a special feature for Dalhousie University. If `FDURLIMIT > 0` then `FLOW_TIMEOUT` is overwritten if `FURLIMIT` seconds are reached.

2.9.17 HASHFACTOR

A factor to be multiplied with the `HASHTABLE_SIZE` and `HASHCHAINTABLE_SIZE` described below. It facilitates the correct setting of the hash space. Moreover, if T2 runs out of hash it will give an upper estimate the user can choose for `HASHFACTOR`. Set it to this value, recompile and rerun T2. This constant is superseeded by the `-f` option.

2.9.18 HASHTABLE_SIZE

The number of buckets in the hash table. As a separate chaining hashing method is used, this value does not denote the amount of elements the hash table is able to manage! The larger, the less likely are hash collisions. The current default value is 2^{18}. Its value should be selected at least two times larger as the value of `HASHCHAINTABLE_SIZE` discussed in the following chapter.

2.9.19 HASHCHAINTABLE_SIZE

Specifies the amount of flows the main hash table is able to manage. The default value is 2^{19}, so roughly half the size of `HASHTABLE_SIZE`. T2 supplies information about the hash space in memory in: `Max number of IPv4 flows in memory: 113244 (50.220%)`. Together with the amount of traffic already processed the total value can be computed. An example is given in Figure 1.

2.9.20 HASH_AUTOPILOT

Default 1. Avoids overrun of main hash, flushes oldest flow on every flowInsert if hashmap is full. 0 disables hash overrun protection. If speed is an issue avoid overruns by invoking T2 with a laged -f option value, as T2 recommends.

2.9.21 Aggregation Mode

The aggregation mode enables the user to confine certain IP, port or protocol ranges into a single flow. The variable `AGGREGATIONFLAG` in tranalyzer.h defines a bit field which enables specific aggregation modes according to the six tuple values listed below.

Aggregation Flag	Value
`L4PROT`	`0x01`
`DSTPORT`	`0x02`
`SRCPORT`	`0x04`
`DSTIP`	`0x08`
`SRCIP`	`0x10`
`VLANID`	`0x20`
`SUBNET`	`0x80`

If a certain aggregation mode is enabled the following variables in tranalyzer.h define the aggregation range.

Aggregation Flag	**Type**	**Description**
`SRCIP4CMSK`	`uint8_t`	src IPv4 aggregation CIDR mask
`DSTIP4CMSK`	`uint8_t`	dst IPv4 aggregation CIDR mask
`SRCIP6CMSK`	`uint8_t`	src IPv6 aggregation CIDR mask
`DSTIP6CMSK`	`uint8_t`	dst IPv6 aggregation CIDR mask
`SRCPORTLW`	`uint16_t`	src port lower bound
`SRCPORTHW`	`uint16_t`	src port upper bound
`DSTPORTLW`	`uint16_t`	dst port lower bound
`DSTPORTHW`	`uint16_t`	dst port upper bound

2.10 bin2txt.h

Name	**Default**	**Description**
`HEX_CAPITAL`	0	hex output: 0: lower case; 1: upper case
`IP4_FORMAT`	0	IPv4 addresses representation: 0: normal, 1: normalized (padded with zeros), 2: one 32-bits hex number 3: one 32-bits unsigned number
`IP6_FORMAT`	0	IPv6 addresses representation: 0: compressed, 1: uncompressed, 2: one 128-bits hex number, 3: two 64-bits hex numbers
`MAC_FORMAT`	0	MAC addresses representation: 0: normal (edit `MAC_SEP` to change the separator), 1: one 64-bits hex number,
`MAC_SEP`	`":"`	Separator to use in MAC addresses: `11:22:33:44:55:66`
`TFS_EXTENDED_HEADER`	0	Extended header in flow file
`B2T_TIME_IN_MICRO_SECS`	1	Time precision: 0: nanosecs, 1: microsecs
`TFS_NC_TYPE`	1	Types in header file: 0: numbers, 1: C types
`TFS_SAN_UTF8`	1	Activates the UTF-8 sanitizer for strings
`B2T_TIMESTR`	0	Print unix timestamps as human readable dates
`HDR_CHR`	`"%"`	start character(s) of comments
`SEP_CHR`	`"\t"`	column separator in the flow file `";"`, `"."`, `"_"` and `"\""` should not be used
`JSON_KEEP_EMPTY`	0	Whether or not to output empty fields
`JSON_PRETTY`	0	Whether to add spaces to make the output more readable

2.11 outputBuffer.h

Name	**Default**	**Description**	**Flags**
`BUF_DATA_SHFT`	0	Adds for each binary output record the length and shifts the record by *n* `uint32_t` words to the right	

Name	Default	Description	Flags
		(see `binSink` and `socketSink` plugin)	
`OUTBUF_AUTOPILOT`	1	Automatically increase the output buffer when required	
`OUTBUF_MAXSIZE_F`	5	Maximal factor to increase the output buffer size to	`OUTBUF_AUTOPILOT=1`

2.12 Tranalyzer2 Output

As stated before, the functionality and output of Tranalyzer2 is defined by the activated plugins. Basically, there are two ways a plugin can generate output. First, it can generate its own output file and write any arbitrary content into any stream. The second way is called standard output or per-flow output. After flow termination Tranalyzer2 provides an output buffer and appends the direction of the flow to it. For example, in case of textual output, an `"A"` flow is normally followed by a `"B"` flow or if the `"B"` flow does not exist it is followed by the next `"A"` flow. Then, the output buffer is passed to the plugins providing their per-flow output. Finally the buffer is sent to the activated output plugins. This process repeats itself for the `"B"` flow. For detailed explanation about the functionality of the output plugins refer to the section plugins.

2.12.1 Hierarchical Ordering of Numerical or Text Output

Tranalyzer2 provides a hierarchical ordering of each output. Each plugin controls the:

- volume of its output
- number of values or bins
- hierarchical ordering of the data
- repetition of data substructures

Thus, complex structures such as lists or matrices can be presented in a single line.
The following sample of text output shows the hierarchical ordering for four data outputs, separated by tabulators:

```
A    0.3   2.0_3.4_2.1    2;4;2;1    (1_2_9)_(1_3_1)_(7_5_3)_(2_3_7)
```

The `A` indicates the direction of the flow; in this case it is the initial flow. The next number denotes a singular descriptive statistical result. Output number two consists of three values separated by "_" characters. Output number three consists of one value, that can be repeated, indicated by the character ";". Output number four is a more complex example: It consists of four values containing three subvalues indicated by the braces. This could be interpreted as a matrix of size 4×3.

2.13 Final Report

Standard configuration of Tranalyzer2 produces a statistical report to *stdout* about timing, packets, protocol encapsulation type, average bandwidth, dump length, etc. A sample report including some current protocol relevant warnings is depicted in the figure below. Warnings are not fatal hence are listed at the end of the statistical report when Tranalyzer2 terminates naturally. The *Average total Bandwidth* estimation refers to the processed bandwidth during the data acquisition process. It is only equivalent to the actual bandwidth if the total packet length including all encapsulations is not truncated and all traffic is IP. The *Average IP Traffic Bandwidth* is an estimate comprising all IP traffic actually present on the wire. Plugins can report extra information when `PLUGIN_REPORT` is activated. This report can be saved in a file, by using one of the following command:

```
tranalyzer -r file.pcap -w out -l (See Section 2.4.8)
tranalyzer -r file.pcap -w out | tee out_stdout.txt
tranalyzer -r file.pcap -w out > out_stdout.txt
```

Both commands will create a file `out_stdout.txt` containing the report. The only difference between those two commands is that the first one still outputs the report to stdout.

Fatal errors regarding the invocation, configuration and operation of Tranalyzer2 are printed to *stderr* after the plugins are loaded, thus before the processing is activated, see the *Hash table error* example in Listing 1. These errors terminate Tranalyzer2 immediately and are located before the final statistical report as being indicated by the *"Shutting down..."* key phrase. If the final report is to be used in a following script a pipe can be appended and certain lines can be filtered using grep or awk.

```
$ ./tranalyzer -r ~/data/knoedel.pcap -w ~/results/
================================================================================
Tranalyzer 0.8.2 (Anteater), Tarantula. PID: 16298
================================================================================
[INF] Creating flows for L2, IPv4, IPv6
Active plugins:
    01: protoStats, 0.8.2
    02: basicFlow, 0.8.2
    03: macRecorder, 0.8.2
    04: portClassifier, 0.8.2
    05: basicStats, 0.8.3
    06: tcpFlags, 0.8.2
    07: tcpStates, 0.8.2
    08: icmpDecode, 0.8.2
    09: dnsDecode, 0.8.3
    10: httpSniffer, 0.8.2
    11: connStat, 0.8.2
    12: txtSink, 0.8.2
[INF] basicFlow: IPv4 Ver: 3, Rev: 20190114, Range Mode: 0, subnet ranges loaded: 2821502 (2.82 M)
[INF] basicFlow: IPv6 Ver: 3, Rev: 20190114, Range Mode: 0, subnet ranges loaded: 36123 (36.12 K)
Processing file: /home/user/data/knoedel.pcap
Link layer type: Ethernet [EN10MB/1]
Dump start: 1291753225.446732 sec (Tue 07 Dec 2010 20:20:25 GMT)
[WRN] snapL2Length: 1550 - snapL3Length: 1484 - IP length in header: 1492
[WRN] Hash Autopilot: main HashMap full: flushing 1 oldest flow(s)
[INF] Hash Autopilot: Fix: Invoke Tranalyzer with '-f 5'
Dump stop : 1291753452.373884 sec (Tue 07 Dec 2010 20:24:12 GMT)
Total dump duration: 226.927152 sec (3m 46s)
Finished processing. Elapsed time: 284.541828 sec (4m 44s)
Finished unloading flow memory. Time: 295.292141 sec (4m 55s)
Percentage completed: 100.00%
Number of processed packets: 53982409 (53.98 M)
Number of processed bytes: 42085954664 (42.09 G)
Number of raw bytes: 42101578296 (42.10 G)
Number of pcap bytes: 42949673232 (42.95 G)
Number of IPv4 packets: 53768016 (53.77 M) [99.60%]
Number of IPv6 packets: 214108 (214.11 K) [0.40%]
Number of A packets: 31005784 (31.01 M) [57.44%]
Number of B packets: 22976625 (22.98 M) [42.56%]
Number of A bytes: 14212871985 (14.21 G) [33.77%]
Number of B bytes: 27873082679 (27.87 G) [66.23%]
Average A packet load: 458.39
Average B packet load: 1213.11 (1.21 K)
--------------------------------------------------------------------------------
```

```
basicStats: Biggest Talker: 92.122.216.218: 154922 (154.92 K) [0.29%] packets
basicStats: Biggest Talker: 92.122.216.218: 224224088 (224.22 M) [0.53%] bytes
tcpFlags: Aggregated IP anomaly flags : 0x3d6e
tcpFlags: Aggregated TCP anomaly flags: 0xfe07
tcpFlags: Number of TCP scans, succ scans, retries: 175001 (175.00 K), 29259 (29.26 K), 15782
    (15.78 K)
tcpFlags: Number WinSz below 1: 162725 (162.72 K) [0.38%]
tcpStates: Aggregated anomaly flags: 0xdf
icmpDecode: Number of ICMP echo request packets: 14979 (14.98 K) [12.00%]
icmpDecode: Number of ICMP echo reply packets: 2690 (2.69 K) [2.16%]
icmpDecode: ICMP echo reply / request ratio: 0.18
icmpDecode: Number of ICMPv6 echo request packets: 1440 (1.44 K) [9.63%]
icmpDecode: Number of ICMPv6 echo reply packets: 951 [6.36%]
icmpDecode: ICMPv6 echo reply / request ratio: 0.66
dnsDecode: Number of DNS packets: 237597 (237.60 K) [0.44%]
dnsDecode: Number of DNS Q packets: 125260 (125.26 K) [52.72%]
dnsDecode: Number of DNS R packets: 112337 (112.34 K) [47.28%]
dnsDecode: Aggregated status: 0xe72f
httpSniffer: Number of HTTP packets: 36614826 (36.61 M) [67.83%]
httpSniffer: Number of HTTP GET  requests: 426825 (426.82 K) [1.17%]
httpSniffer: Number of HTTP POST requests: 39134 (39.13 K) [0.11%]
httpSniffer: HTTP GET/POST ratio: 10.91
httpSniffer: Aggregated status flags : 0x003f
httpSniffer: Aggregated anomaly flags: 0x5143
httpSniffer: Aggregated content flags: 0x007a
httpSniffer: Aggregated mime type    : 0x80ef
connStat: Max unique number of IP source connections: 275532 (275.53 K)
connStat: Max unique number of IP destination connections: 301252 (301.25 K)
connStat: Max unique number of IP source/destination connections: 1242 (1.24 K)
connStat: Max unique number of source IP / destination port connections: 1521 (1.52 K)
connStat: prtcon/sdcon, prtcon/scon: 1.224638, 0.005520
connStat: Source IP with max connections: X.Y.Z.U: 1515 (1.51 K) connections
connStat: Destination IP with max connections: V.W.A.B: 3241 (3.24 K) connections
--------------------------------------------------------------------------------
Headers count: min: 4, max: 14, average: 7.10
Max VLAN header count: 1
Max MPLS header count: 2
Number of LLC packets: 285 [0.00%]
Number of GRE packets: 27670 (27.67 K) [0.05%]
Number of Teredo packets: 213877 (213.88 K) [0.40%]
Number of AYIYA packets: 231 [0.00%]
Number of IGMP packets: 401 [0.00%]
Number of ICMP packets: 124800 (124.80 K) [0.23%]
Number of ICMPv6 packets: 14946 (14.95 K) [0.03%]
Number of TCP packets: 43273340 (43.27 M) [80.16%]
Number of UDP packets: 10311931 (10.31 M) [19.10%]
Number of IPv4 fragmented packets: 19155 (19.16 K) [0.04%]
Number of IPv6 fragmented packets: 9950 (9.95 K) [4.65%]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of processed   flows: 1439153 (1.44 M)
Number of processed A flows: 1209728 (1.21 M) [84.06%]
Number of processed B flows: 229425 (229.43 K) [15.94%]
Number of request     flows: 930935 (930.93 K) [64.69%]
Number of reply       flows: 508218 (508.22 K) [35.31%]
Total   A/B    flow asymmetry: 0.68
Total req/rply flow asymmetry: 0.29
Number of processed   packets/flows: 37.51
Number of processed A packets/flows: 25.63
Number of processed B packets/flows: 100.15
Number of processed total packets/s: 237884.31 (237.88 K)
```

```
Number of processed A+B packets/s: 237884.31 (237.88 K)
Number of processed A   packets/s: 136633.20 (136.63 K)
Number of processed   B packets/s: 101251.10 (101.25 K)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of average processed flows/s: 6341.92 (6.34 K)
Average full raw bandwidth: 1484232320 b/s (1.48 Gb/s)
Average snapped bandwidth : 1483681536 b/s (1.48 Gb/s)
Average full bandwidth : 1483899904 b/s (1.48 Gb/s)
Max number of flows in memory: 262144 (262.14 K) [100.00%]
Number of flows terminated by autopilot: 816044 (816.04 K) [56.70%]
Memory usage: 2.73 GB [4.05%]
Aggregate flow status: 0x01003cfad298fb04
[WRN] L3 SnapLength < Length in IP header
[WRN] L4 header snapped
[WRN] Consecutive duplicate IP ID
[WRN] IPv4/6 fragmentation header packet missing
[WRN] IPv4/6 packet fragmentation sequence not finished
[INF] IPv4
[INF] IPv6
[INF] IPv4/6 fragmentation
[INF] VLAN encapsulation
[INF] MPLS encapsulation
[INF] L2TP encapsulation
[INF] PPP/HDLC encapsulation
[INF] GRE encapsulation
[INF] AYIYA tunnel
[INF] Teredo tunnel
[INF] CAPWAP/LWAPP tunnel
[INF] SSDP/UPnP flows
[INF] Ethernet flows
[INF] SIP/RTP flows
[INF] Authentication Header (AH)
[INF] Encapsulating Security Payload (ESP)
[INF] TOR addresses
```

Listing 1: *A sample Tranalyzer2 final report including encapsulation warning, Hash Autopilot engagement when hash table full*

T2 runs in IPv4 mode, but warns the user that there is IPv6 encapsulated. Note that the new Hash Autopilot warns you when the main hash map is full. T2 then removes the oldest Flow and continues processing your pcap. To avoid that, run T2 again, but this time, use the `-f 5` option as indicated in the warning message: `[INF] Hash Autopilot: Fix: Invoke Tranalyzer with '-f 5'`
`t2 -r ~/wurst/data/knoedel.pcap -w ~/results -f 5` or just let it run to the finish.

2.14 Monitoring Modes During Runtime

If debugging is deactivated or the verbose level is zero (see Section 2.9), Tranalyzer2 prints no status information or end report. Interrupt signal has been introduced to force intermediate status information to `stdout`. Appropriate Unix commands and their effect are listed below:

Command	Description
`kill -USR1 PID`	T2 sends configured monitoring report to stdout
`kill -USR2 PID`	T2 toggles between on demand and continuous monitoring operation

The script `t2stat` has the same function as `kill -USR1 PID`. An example of a typical signal requested report

(MACHINE_REPORT=0) is shown in Listing 2.

```
                                    @      @
                                     |    |
===============================vVv==(a     a)==vVv===============================
=====================================\     /=====================================
======================================\   /======================================
                                       oo
USR1 A type report: Tranalyzer 0.8.2 (Anteater), Tarantula. PID: 16355
PCAP time: 1291753261.106203 sec (Tue 07 Dec 2010 20:21:01 GMT)
PCAP duration: 35.659471 sec
Time: 1546607860.487896 sec (Fri 04 Jan 2019 14:17:40 CET)
Elapsed time: 21.399345 sec
Total bytes to process: 42949673232 (42.95 G)
Percentage completed: 16.01%
Total bytes processed so far: 6875527168 (6.88 G)
Remaining time: 112.276935 sec (1m 52s)
ETF: 1546607972.764831 sec (Fri 04 Jan 2019 14:19:32 CET)
Number of processed packets: 8576716 (8.58 M)
Number of processed bytes: 6738299436 (6.74 G)
Number of raw bytes: 6740587316 (6.74 G)
Number of IPv4 packets: 8543638 (8.54 M) [99.61%]
Number of IPv6 packets: 33035 (33.03 K) [0.39%]
Number of A packets: 4916815 (4.92 M) [57.33%]
Number of B packets: 3659901 (3.66 M) [42.67%]
Number of A bytes: 2267361054 (2.27 G) [33.65%]
Number of B bytes: 4470938382 (4.47 G) [66.35%]
Average A packet load: 461.14
Average B packet load: 1221.60 (1.22 K)
---------------------------------------------------------------------------------
tcpFlags: Number of TCP scans, succ scans, retries: 9533, 4061, 2125
icmpDecode: Number of ICMP echo request packets: 2610 (2.61 K) [11.39%]
icmpDecode: Number of ICMP echo reply packets: 703 [3.07%]
dnsDecode: Number of DNS packets: 37156 (37.16 K) [0.43%]
dnsDecode: Number of DNS Q packets: 18735 (18.73 K) [50.42%]
dnsDecode: Number of DNS R packets: 18421 (18.42 K) [49.58%]
dnsDecode: Aggregated status: 0x0201
httpSniffer: Number of HTTP packets: 5833252 (5.83 M) [68.01%]
connStat: Max unique number of IP source connections: 61971 (61.97 K)
connStat: Max unique number of IP destination connections: 67895 (67.89 K)
connStat: Max unique number of IP source/destination connections: 712
connStat: Max unique number of source IP / destination port connections: 712
connStat: IP prtcon/sdcon, prtcon/scon: 1.000000, 0.011489
---------------------------------------------------------------------------------
Headers count: min: 4, max: 13, average: 6.95
Max VLAN header count: 1
Max MPLS header count: 2
Number of LLC packets: 43 [0.00%]
Number of GRE packets: 3677 (3.68 K) [0.04%]
Number of Teredo packets: 33001 (33.00 K) [0.38%]
Number of AYIYA packets: 34 [0.00%]
Number of IGMP packets: 58 [0.00%]
Number of ICMP packets: 20491 (20.49 K) [0.24%]
Number of ICMPv6 packets: 2426 (2.43 K) [0.03%]
Number of TCP packets: 6907067 (6.91 M) [80.53%]
Number of UDP packets: 1608944 (1.61 M) [18.76%]
Number of IPv4 fragmented packets: 2511 (2.51 K) [0.03%]
Number of IPv6 fragmented packets: 540 [1.63%]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
Number of processed   flows: 270826 (270.83 K)
Number of processed A flows: 228096 (228.10 K) [84.22%]
Number of processed B flows: 42730 (42.73 K) [15.78%]
Number of request     flows: 224453 (224.45 K) [82.88%]
Number of reply       flows: 46373 (46.37 K) [17.12%]
Total   A/B    flow asymmetry: 0.68
Total req/rply flow asymmetry: 0.66
Number of processed   packets/flows: 31.67
Number of processed A packets/flows: 21.56
Number of processed B packets/flows: 85.65
Number of processed total packets/s: 240517.21 (240.52 K)
Number of processed A+B packets/s: 240517.21 (240.52 K)
Number of processed A   packets/s: 137882.45 (137.88 K)
Number of processed   B packets/s: 102634.76 (102.63 K)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of average processed flows/s: 7594.78 (7.59 K)
Average full raw bandwidth: 1512212608 b/s (1.51 Gb/s)
Average snapped bandwidth : 1511699328 b/s (1.51 Gb/s)
Average full bandwidth : 1511878912 b/s (1.51 Gb/s)
Fill size of main hash map: 236513 [90.22%]
Max number of flows in memory: 236517 (236.52 K) [90.22%]
Memory usage: 2.49 GB [3.69%]
Aggregate flow status: 0x01003872d298fb04
[WRN] L3 SnapLength < Length in IP header
[WRN] Consecutive duplicate IP ID
[WRN] IPv4/6 fragmentation header packet missing
[INF] IPv4
[INF] IPv6
[INF] IPv4/6 fragmentation
[INF] VLAN encapsulation
[INF] MPLS encapsulation
[INF] L2TP encapsulation
[INF] PPP/HDLC encapsulation
[INF] GRE encapsulation
[INF] AYIYA tunnel
[INF] Teredo tunnel
[INF] CAPWAP/LWAPP tunnel
[INF] SSDP/UPnP flows
[INF] Ethernet flows
[INF] SIP/RTP flows
[INF] Authentication Header (AH)
[INF] Encapsulating Security Payload (ESP)
[INF] TOR addresses
================================================================================
```

Listing 2: *A sample Tranalyzer2 human readable report aggregate mode*

Listing 3 illustrates the output of the header line and subsequent data lines generated when `MACHINE_REPORT=1`.

```
%RepTyp Time    Dur memUsg[KB]  fillSzHashMap   Flws    AFlws   BFlws   SIP DIP SDIP    Prts
    Pkts    APkts   BPkts   V4Pkts  V6Pkts  VxPkts  Byts    AByts   BByts   ARPPkts RARPPkts
    ICMPPkts    EchoReq EchoRep DnsPkts DnsQPkts    DnsRPkts    HttpPkts    FrgV4Pkts   FrgV6Pkts
       Alrms   TCPScn  TCPSScn TcpRtry RawBndWdth  Fave    GlblWrn ICMP    IGMP    TCP UDP SCTP
        0x0042  0x00fe  0x0806  0x0800  0x86dd
USR1MR_A    1175364020.458606   103.877184  1856384     6456055 518079  360556  157523  92140
    153825  765 1706    9453374 5485249 3967298 9452802 0   0   680258574   2271454049  3288136123
      0   0   492113898   1244    0   0   0   3309    20599   0   0   26244   22467   14833
    428175.000  2.230065    0x02030052  49211   102 8186447 1169051 0   284 0   0   9452802 0
USR1MR_A    1175364043.564798   126.983376  2030752     6945643 680946  462230  218716  118181
    183651  1164    3626    13104550    7235588 5867866 13103799    0   0   944828393   3007472308
      4847491640  0   066918  5110    1695    0   0   0   4738    28844   0   0   33600   32326
```

```
    23344   494874.688  3.115120    0x02030052  66918   130 11364263    1611189 0   372 0   0
    13103799    0
USR1MR_A    1175364072.842855   156.261433  2184732     7103403 860105  573862  286243  148357
    217272  1164    3626    17716953    9443884 8271657 17715947    0   0   1278800701  3927484585
      6829832070  0   086334  6700    2226    0   0   0   6572    38940   0   0   48853   45035
    34063   550743.562  3.115120    0x02030052  86334   177 15394866    2158059 0   492 0   0
    17715947    0
```

Listing 3: *A sample Tranalyzer2 machine report aggregate mode*

2.14.1 Configuration for Monitoring Mode

To enable monitoring mode, configure Tranalyzer as follows:

main.h	tranalyzer.h
`#define MONINTPSYNC 1`	`#define VERBOSE 0`
`#define MONINTTMPCP 1`	`#define DIFF_REPORT 1`
`#define MONINTTHRD 1`	`#define MACHINE_REPORT 1`

The following plugins contribute to the output:

- `basicStats`
- `connStat`
- `dnsBCmp`
- `dnsDecode`
- `httpBCmp`
- `icmpDecode`
- `protoStats`

The generated output is illustrated in Figure 3. The columns are as follows:

1. RepType
2. Time
3. Dur
4. memUsage[KB]
5. fillSzHashMap
6. Flows
7. AFlows
8. BFlows
9. SIP
10. DIP
11. SDIP
12. Prts
13. Pkts
14. APkts
15. BPkts
16. V4Pkts
17. V6Pkts
18. VxPkts
19. Byts
20. AByts
21. BByts
22. ARPPkts
23. RARPPkts
24. ICMPPkts
25. EchoReq
26. EchoRep
27. DnsPkts
28. DnsQPkts
29. DnsRPkts
30. HttpPkts
31. FrgV4Pkts
32. FrgV6Pkts
33. Alrms
34. TCPScn
35. TCPSScn
36. TcpRtry
37. RawBndWdth
38. Fave
39. GlblWrn

When capturing from a live interface, the following three columns are output (between `Dur` and `memUsage[KB]`):

- PktsRec
- PktsDrp
- IfDrp

2.14.2 Monitoring Mode to syslog

In order to send monitoring info to a syslog server T2 must be configured in machine mode as indicated above. Then the output has to be piped into the following script:

```
t2 -D ... -w ... | awk -F"\t" ‘{ print "<25> ", strftime("%b %d %T"), "Monitoring: " $0; }’ | \
        nc -u w.x.y.z 514
```

Netcat will send it to the syslog server at address `w.x.y.z`. Specific columns from the monitoring output can be selected in the awk script.

2.14.3 RRD Graphing of Monitoring Output

The monitoring output can be stored in a RRD database using the `rrdmonitor` script. To start creating a RRD database, launch Tranalyzer2 (in monitoring mode) as follows:

```
./tranalyzer2/src/tranalyzer -r file.pcap | ./scripts/rrdmonitor
```

Or for monitoring from a live interface:

```
sudo ./tranalyzer2/src/tranalyzer -i eth0 | ./scripts/rrdmonitor
```

Plots for the various fields can then be generated using the `rrdplot` script. To specify intervals, use s (seconds), m (minutes), h (hour), d (day), w (week), mo (month), y (year). For example, to plot the data from the last two weeks, use `-i 2w` or `-s -2w`.

An example graph is depicted in Figure 3.

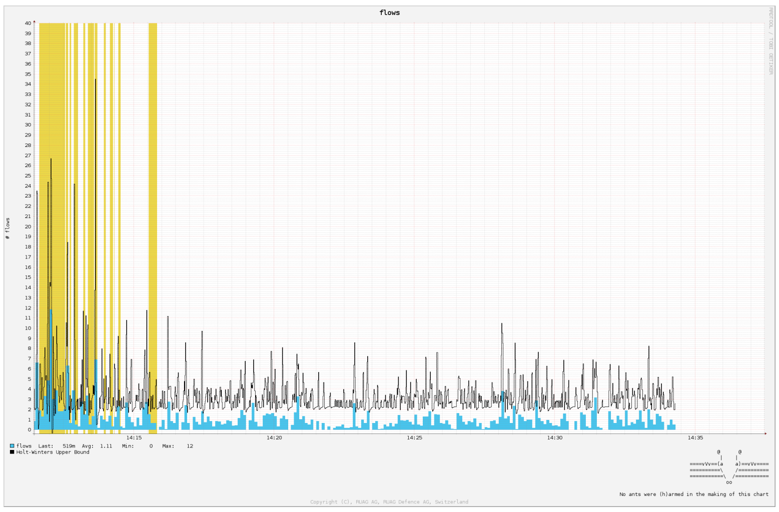


Figure 3: *T2 monitoring using RRD*

2.15 Cancellation of the Sniffing Process

Processing of a pcap file stops upon end of file. In case of live capture from an interface Tranalyzer2 stops upon `CTRL+C` interrupt or a `kill -9 PID` signal. The disconnection of the interface cable will stop Tranalyzer2 also after a timeout of 182 seconds. The console based `CTRL+C` interrupt does not immediately terminate the program to avoid corrupted entries in the output files. It stops creating additional flows and finishes only currently active flows. Note that waiting the termination of active flow depends on the activity or the lifetime of a connection and can take a very long time. In order to mitigate that problem the user can issue the `CTRL+C` for `GI_TERM_THRESHOLD` times to immediately terminate the program.