Tranalyzer2

scripts

Various Scripts and Utilities

Tranalyzer Development Team

Contents

1 scripts

This section describes various scripts and utilities for Tranalyzer. For a complete list of options, use the scripts `-h` option.

1.1 b64ex

Extracts all HTTP, EMAIL, FTP, TFTP, etc base 64 encoded content extracted from T2. To produce a list of files containing base64 use `grep` as indicated below:

- `grep "base64" /tmp/SMTPFILE/*`
- `./b64ex /tmp/SMTPFILES/file@wurst.ch_0_1223`

1.2 fpsGplt

Transforms the output of the `nFrstPkts` plugin signal output to gnuplot or t2plot format for encrypted traffic mining purposes. It generates an output file: `flowfile_nps.txt` containing the processed PL signal according to `nFrstPkts` plugin configuration.

```
$ fpsGplt -h
Usage:
    fpsGplt [OPTION...] <FILE_flows.txt>

Optional arguments:
    -f findex        Flow index to extract [default: all flows]
    -d A|B           Flow direction: A or B only [default: A and B]
    -t               No time, but counts on x axis [default: time on x axis]
    -i               Invert B flow PL
    -s               Time sorted ascending
    -p s             Sample sorted signal with smplIAT in [s]; f = 1/smplIAT
    -e s             Time for each PL pulse edge in [s]
    -j               Calculate the jumps in IAT and report appropriate values
                     for MINIAT(S/U)
    -h, --help       Show this help, then exit
```

If `-f` is omitted all flows will be included. If `-d` is omitted both flow directions will be processed. `-t` removes the timestamp and replaces it with an integer count. `-i` inverts the `B` flow signal to produce a symmetrical signal. `-p` samples the sorted signal with the IAT in seconds resp. frequency you deem necessary and `-e` defines the pulse flank in seconds. `-j` calculates the jumps in IAT to allows the user to choose an appropriate `MINIAT(S/U)` in `nFrstPkts` plugin.

1.3 gpq3x

Use this script to create 3D waterfall plot. Was originally designed for the `centrality` plugin:

```
cat FILE_centrality | ./gpq3x
```

The script can be configured through the command line. For a full list of options, run `./gpq3x -help`

1.4 osStat

Counts the number of hosts of each operating system (OS) in a PCAP file. In addition, a file with suffix `_IP_OS.txt` mapping every IP to its OS is created. This script uses `p0f` which requires a fingerprints file (`p0f.fp`), the location of which can be specified using the `-f` option. Version 2 looks first in the current directory, then in `/etc/p0f`. Version 3 looks only in the current directory.

- list all the options: `osStat --help`
- top 10 OS: `osStat file.pcap -n 10`
- bottom 5 OS: `osStat file.pcap -n -5`

1.5 protStat

The `protStat` script can be used to sort the `PREFIX_protocols.txt` file (generated by the `protoStats` plugin) or the `PREFIX_nDPI.txt` file (generated by the `nDPI` plugin) for the most or least occurring protocols (in terms of number of packets or bytes). In addition, it can also sort the `PREFIX_icmpStats.txt` and `PREFIX_igmpStats.txt` files (generated by the `icmpDecode` and `igmpDecode` plugins respectively). It can output the top or bottom *N* protocols or only those with at least a given percentage:

- list all the options: `protStat --help`
- for better readability, use `protStat` with `tcol`: `protStat ... | tcol`
- sorted list of protocols (by packets): `protStat PREFIX_protocols.txt`
- sorted list of protocols (by bytes): `protStat PREFIX_protocols.txt -b`
- top 10 protocols (by packets): `protStat PREFIX_protocols.txt -n 10`
- bottom 5 protocols (by bytes): `protStat PREFIX_protocols.txt -n -5 -b`
- protocols with packets percentage greater than 20%: `protStat PREFIX_protocols.txt -p 20`
- protocols with bytes percentage smaller than 5%: `protStat PREFIX_protocols.txt -b -p -5`
- TCP and UDP statistics only: `protStat PREFIX_protocols.txt -udp -tcp`

1.6 statGplt

Transforms 2/3D statistics output from `pktSIATHisto` plugin to gnuplot or t2plot format for encrypted traffic mining purposes.

1.7 t2_aliases

Set of aliases for Tranalyzer.

1.7.1 Description

`t2_aliases` defines the following aliases, functions and variables:

T2HOME
Variable pointing to the root folder of Tranalyzer, e.g., `cd $T2HOME`.

T2PLHOME
Variable pointing to the root folder of Tranalyzer plugins, e.g., `cd $T2PLHOME`. In addition, every plugin can be accessed by typing its name instead of its full path. For example to access `tcpFlags` home folder, `tcpFlags` can be used instead of `cd $T2PLHOME/tcpFlags` or `cd $T2HOME/plugins/tcpFlags`.

tran
Shortcut to access `$T2HOME`, e.g., `tran`

tranpl
Shortcut to access `$T2PLHOME`, e.g., `tranpl`

.tran
Shortcut to access `$HOME/.tranalyzer/plugins`, e.g., `.tran`

awkf
Configures `awk` to use tabs, i.e., '`\t`' as input and output separator (prevents issue with repetitive values), e.g., `awkf '{ print $4 }' file_flows.txt`

tawk
Shortcut to run tawk from anywhere, e.g., `tawk`

tcol
Displays columns with minimum width, e.g., `tcol file_flows.txt`.

lsx
Displays columns with fixed width (default: 40), e.g., `lsx file_flows.txt` or `lsx 45 file_flows.txt`.
Note that ZSH already defines a `lsx` alias, therefore if using ZSH this command will **NOT** be installed. To have it installed, add the following line to your `~/.zshrc` file: `unalias lsx`

sortu
Sort rows and count the number of times a given row appears, then sort by the most occurring rows. (Alias for `sort | uniq -c | sort -rn`). Useful, e.g., to analyze the most occurring user-agents: `tawk '{ print $httpUsrAg }' FILE_flows.txt | sortu`

sortup
Same as `sortu`, but display the relative percentage instead of the absolute count. e.g., to analyze the most occurring user-agents: `tawk '{ print $httpUsrAg }' FILE_flows.txt | sortup`

t2
Shortcut to run Tranalyzer from anywhere, e.g., `t2 -r file.pcap -w out`

gt2
Shortcut to run Tranalyzer in `gdb` (Linux) or `lldb` (macOS) from anywhere, e.g., `gt2 -r file.pcap -w out`

st2
Shortcut to run Tranalyzer with sudo, e.g., `st2 -i eth0 -w out`

tranalyzer
Shortcut to run Tranalyzer from anywhere, e.g., `tranalyzer -r file.pcap -w out`

fextractor
Shortcut to run `fextractor` from anywhere, e.g., `fextractor -r file_flows.xer 1234`

fpsGplt
Shortcut to run `fpsGplt` from anywhere, e.g., `fpsGplt file_flows.txt`

protStat
Shortcut to run `protStat` from anywhere, e.g., `protStat file_protocols.txt`

statGplt
Shortcut to run `statGplt` from anywhere, e.g., `statGplt file_flows.txt`

t2b2t
Shortcut to run t2b2t from anywhere, e.g., `t2b2t -r file_flows.bin -w file_flows.txt`.

t2build
Function to build Tranalyzer and the plugins from anywhere, e.g., `t2build tcpFlags`. Use `<tab>` to list the available plugins and complete names. Use `t2build -h` for a full list of options.

t2caplist
Shortcut to run t2caplist from anywhere, e.g., `t2caplist`

t2conf
Shortcut to run t2conf from anywhere, e.g., `t2conf --gui`

t2dmon
Shortcut to run t2dmon from anywhere, e.g., `t2dmon dumps/`

t2doc
Shortcut to run t2doc from anywhere, e.g., `t2doc tranalyzer2`

t2docker
Shortcut to run t2docker from anywhere, e.g., `t2docker -r file.pcap`

t2dpdk
Shortcut to run t2dpdk from anywhere, e.g., `t2dpdk -N 4 -i 0000:04:00.0`.

t2flowstat
Shortcut to run t2flowstat from anywhere, e.g., `t2flowstat file_flows.txt -c pktsSnt -s 1 -m 9000 -0`

t2fm
Shortcut to run t2fm from anywhere, e.g., `t2fm -r file.pcap`

t2fuzz
Shortcut to run t2fuzz from anywhere, e.g., `t2fuzz file.pcap`

t2locate
Shortcut to run t2locate from anywhere, e.g., `t2locate`

t2mmdb
Shortcut to run `t2mmdb` (see `geoip` plugin documentation for more information) from anywhere, e.g., `t2mmdb`

t2netID
Shortcut to run t2netID from anywhere, e.g., `t2netID 0x138020a5`

t2plot
Shortcut to run t2plot from anywhere, e.g., `t2plot file.txt`

t2plugin
Shortcut to run t2plugin from anywhere, e.g., `t2plugin -c pluginName`.

t2rrd
Shortcut to run t2rrd from anywhere, e.g., `t2 -i eth0 | t2rrd -m` or `t2rrd V4Pkts V6Pkts`

t2stat
Shortcut to run t2stat from anywhere, e.g., `t2stat -USR2`

t2test
Shortcut to run `tests/T2Tester.py` from anywhere, e.g., `t2test tranalyzer2`

t2timeline
Shortcut to run t2timeline from anywhere, e.g., `t2timeline file.txt`

t2topcap
Shortcut to run t2topcap from anywhere, e.g., `t2topcap file.pcapng`

t2update
Shortcut to run ./setup.sh –C from anywhere, e.g., `t2update`. This alias checks for the availability of a new version of Tranalyzer and proceed with the installation if requested.

t2viz
Shortcut to run t2viz from anywhere, e.g., `t2viz file.txt`

t2whois
Shortcut to run t2whois from anywhere, e.g., `t2whois 1.2.3.4`

1.7.2 Usage

Those aliases can be activated using either one of the following methods:

1. Append the content of this file to `~/.bash_aliases` or `~/.bashrc`
2. Append the following line to `~/.bashrc` (make sure to replace `$T2HOME` with the actual path, e.g., `$HOME/tranalyzer2-0.9.3`):

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

1.7.3 Known Bugs and Limitations

ZSH already defines a `lsx` alias, therefore if using ZSH this command will **NOT** be installed. To have it installed, add the following line to your `~/.zshrc` file: `unalias lsx`

1.8 t2alive

In order to monitor the status of T2, the t2alive script sends syslog messages to server defined by the user whenever the status of T2 changes. It acquires the PID of the T2 process and transmits every `REP` seconds a `kill -SYS $pid`. If T2 answers with a corresponding kill command defined in *tranalyzer.h*, s.b., then status is set to alive, otherwise to dead. Only if a status change is detected a syslog message is transmitted. The following constants residing in *tranalyzer.h* govern the functionality of the script:

Name	Default	Description
SERVER	"127.0.0.1"	syslog server IP
PORT	514	syslog server port
FAC	"<25>"	facility code
STATFILE	"/tmp/t2alive.txt"	alive status file
REP	10	T2 test interval [s]

Table 1: *t2alive script configuration*

T2 on the other hand has also to be configured. To preserve simplicity the unused SYS interrupt was abused to respond to the `t2alive` request, hence the monitoring mode depending on USR1 and USR2 can be still functional. Configuration is carried out in *tranalyzer.h* according to the table below:

Name	Default	Description
REPSUP	0	1: activate alive mode
ALVPROG	"t2alive"	name of control program
REPCMDAW	"a=`pgrep ALVPROG`; if [$a]; then kill -USR1 $a; fi"	alive and stall (no packets, looping?)
REPCMDAS	"a=`pgrep ALVPROG`; if [$a]; then kill -USR2 $a; fi"	alive and well (working)

Table 2: *T2 configuration for t2alive mode*

`REPSUP=1` activates the alive mode. If more functionality is requested the `REPCMDAx` constant facilitates the necessary changes. On some Linux distributions the pcap read callback function is not thread safe, thus signals of any kind might

lead to crashes especially when capturing live traffic. Therefore **MONINTTHRD=1** in *main.h* is set by default.
Note that t2alive should be executed in a shell as a standalone script. If executed as a cron job, the while loop and the sleep command has to be removed, as described in the script itself.

1.9 t2b2t

The program `t2b2t` can be used to transform binary Tranalyzer files generated by the `binSink` or `socketSink` plugin into text or json files. The converted files use the same format as the ones generated by the `txtSink` or `jsonSink` plugin.

The program can be found in `$T2HOME/utils/t2b2t` and can be compiled by typing `make`.

The use of the program is straightforward:

- bin→txt: `t2b2t -r FILE_flows.bin -w FILE_flows.txt`
- bin→json: `t2b2t -r FILE_flows.bin -j -w FILE_flows.json`
- bin→compressed txt: `t2b2t -r FILE_flows.bin -c -w FILE_flows.txt.gz`
- bin→compressed json: `t2b2t -r FILE_flows.bin -c -j -w FILE_flows.json.gz`

If the `-w` option is omitted, the destination is inferred from the input file, e.g., the examples above would produce the same output files with or without the `-w` option. Note that `-w -` can be used to output to stdout.
Additionally, the `-n` option can be used **not** to print the name of the columns as the first row.
Try `t2b2t -h` for more information.

1.10 t2caplist

Generates a list of PCAP files with absolute path to use with Tranalyzer `-R` option. If no argument is provided, then lists all the PCAP files in the current directory. If a folder name is given, lists all capture files in the folder. If a list of files is given, list those files. Try `t2caplist -help` for more information.

- `t2caplist > pcap_list.txt`
- `t2caplist ~/dumps/ > pcap_list.txt`
- `t2caplist ~/dumps/testnet*.pcap > pcap_list.txt`

1.11 t2conf

Use `t2conf` to configure, activate and deactivate Tranalyzer plugins:

- To change the value of a configuration flag run:

```
t2conf pluginName -D FLAG_NAME=new_value
```

- To check the value of a configuration flag run:

```
t2conf pluginName -G FLAG_NAME
```

- To list the configuration flags available run:

```
t2conf pluginName -I
```

- To reset a plugin configuration to its default values run:

```
t2conf pluginName --reset
```

- To save a plugin configuration run:

```
t2conf pluginName -g
```

  - Note that the `-g` option accepts a filename:

```
t2conf pluginName -g /where/to/save/file.config
```

- To load plugin configuration run:

```
t2conf pluginName -C /path/to/file.config
```

  - Note that if the default filename is used, `-C auto` can be used instead:

```
t2conf pluginName -C auto
```

- For more details about `t2conf`, run:

```
t2conf --help
```

Alternatively, use `t2conf --gui` option or the `t2plconf` script provided with all the plugins to configure individual plugins as follows:

- `cd $T2PLHOME/pluginName`
- `./t2plconf`
  - Navigate through the different options with the up and down arrows
  - Use the left and right arrows to select an action:
    - `ok`: apply the changes
    - `configure`: edit the selected entry (use the space bar to select a different value)
    - `cancel`: discard the changes
    - `edit`: open the file containing the selected option in `EDITOR` (default: `vim`)
  - Use the space bar to select a different value

A more detailed description of the script can be found in Tranalyzer2 documentation.

1.11.1 Dependencies

The `t2conf` (if the `-gui` option is used) and `t2plconf` scripts require *dialog* (version 1.1-20120703 minimum) and the *vim* editor. The easiest way to install them is to use the `install.sh` script provided (Section 1.11.2). Note that the editor can be changed by exporting the environment variable `EDITOR` as follows: `export EDITOR=/path/to/editor`, e.g., `export EDITOR=/usr/bin/nano` or by setting the `EDITOR` variable at line 9 of the `t2conf` script and at line 90 of the `t2plconf` script.

1.11.2 Installation

The easiest way to install `t2conf` and its dependencies is to use tranalyzer2 `setup.sh` script. Alternatively, the provided `install.sh` script can be used to only install `t2conf`. Run `./install.sh --help` to see what can be installed.

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

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

Where `$T2HOME` is the root folder containing the source code of Tranalyzer2 and its plugins, i.e., where `README.md` is located.

1.11.3 Usage

For a complete list of options use the `-h` option, i.e., `t2conf -h`, or the man page (`man t2conf`).

1.11.4 Patch

`t2conf` can be used to patch Tranalyzer and the plugins (useful to save settings such as hash table size, IPv6, ...).

The format of the patch file is as follows:

- Empty lines and lines starting with `'%'` or `'#'` are ignored
- Filenames are relative to `$T2HOME`
- A line is composed of three or four tabs (not spaces) separated columns:
  - `NAME <tab> newvalue <tab> oldvalue <tab> file`
  - `NAME <tab> newvalue <tab> file`
- `--patch` uses newvalue
- `--rpatch` uses oldvalue[1]

As an example, let us take the value `T2PSKEL_IP` defined in *t2PSkel/src/t2PSkel.h*:

```
#define T2PSKEL_IP   1 // whether or not to output IP (var2)
```

A patch to set this value to 0 would look as follows (where the spaces between the columns are tabs, i.e., `'\t'`):

- `T2PSKEL_IP 0 1 t2PSkel/src/t2PSkel.h`
- `T2PSKEL_IP 0 t2PSkel/src/t2PSkel.h`

1.12 t2dmon

Monitors a folder for new files and creates symbolic links with incrementing indexes. This can be used with the `-D` option when the filenames have either multiple indexes, e.g., date and count, or when the filenames do not possess an index.

1.12.1 Dependencies

This script requires **inotify-tools**:

[1] This option is not valid if the patch has only three columns.

Arch: `sudo pacman -S inotify-tools`

Fedora: `sudo yum install inotify-tools`

Gentoo: `sudo emerge inotify-tools`

Ubuntu: `sudo apt-get install inotify-tools`

1.12.2 Usage

t2dmon works as a daemon and as such, should either be run in the background (the ampersand `'&'` in step 1 below) or on a different terminal.

1. `t2dmon dumps/ -o nudel.pcap &`
2. `tranalyzer -D dumps/nudel.pcap0 -w out`
3. Finally, copy/move the pcap files into the `dumps/` folder.

1.13 t2doc

Access Tranalyzer documentation from anywhere, e.g., `t2doc tcpFlags`. Use `<tab>` to list the available plugins and complete names.

1.14 t2docker

Create and manage Tranalyzer Docker containers.

- Create a Docker container: `t2docker -B t2-latest.tar.gz`
- Download the latest version of T2 and create a Docker container: `t2docker -B latest`
- List existing Tranalyzer Docker containers: `t2docker -ls`
- Save a Docker image: `t2docker -S image-name-or-id`
- Load a Docker image: `t2docker -L t2docker-image.tar[.gz]`
- Get a Shell in a Docker image: `t2docker -X image-name-or-id`
- Run Tranalyzer inside a Docker container: `t2docker -r file.pcap`
- Run another T2 command inside a Docker container: `t2docker tawk -V flowStat`

Note that when running T2 inside a Docker container, the pcap file is copied to a temporary folder (automatically removed or shredded (`--shred` option) after the script has ended). This extra step make sure only the required files are accessible by the container.

1.15 t2dpdk

Run *N* instances of T2 in DPDK multi-process mode. T2 must be compiled with `DPDK_MP=1`:

```
$ t2conf tranalyzer2 -D DPDK_MP=1
$ t2build -R -r -f
$ t2dpdk -N 4 -i 0000:04:00.0
```

1.16 t2flowstat

Calculates statistical distributions of selected columns/flows from a flow file.

1.17 t2fm

Generates a PDF report out of:

- a flow file (`-F` option): `t2fm -F file_flows.txt`
- a live interface (`-i` option): `t2fm -i eth0`
- a PCAP file (`-r` option): `t2fm -r file.pcap`
- a list of PCAP files (`-R` option): `t2fm -R pcap_list.txt`

1.17.1 Required Plugins

- basicFlow
- basicStats
- txtSink

1.17.2 Optional Plugins

- arpDecode
- dnsDecode
- geoip
- httpSniffer
- nDPI
- portClassifier
- pwX
- sshDecode

1.18 t2fuzz

Randomly corrupt a PCAP file and run T2 against it: `t2fuzz file.pcap`.
Try `t2fuzz --help` for more information.

1.19 t2locate

Query location database to acquire information about a city near float point terrestrial coordinates. Build the DB first with the script: `$T2HOME/scripts/t2locate/update_db`, duration ca 30 min.

1.20 t2netID

Decode hexadecimal network IDs. An ID contain information about the country, the organization and whether or not the address is a Tor address. An ID can be decoded as follows: `t2netID 0x138020a5`.
Try `t2netID --help` for more information.

1.21 t2plot

2D/3D plot for Tranalyzer using gnuplot. First row of the input file must be the column names (may start with a '%'). The input file must contain one, two or more columns separated by tabs (\t). Columns to plot can be selected with -o option. Try t2plot --help for more information.

Dependencies: The t2plot script requires **gnuplot**.

Arch: sudo pacman -S gnuplot

Ubuntu: sudo apt-get install gnuplot-qt

macOS: brew install gnuplot --with-qt

Examples:

- tawk '{ print ip2num(shost()), ip2num(dhost()) }' f_flows.txt | t2plot -pt
- tawk '{ print ip2num($srcIP), $timeFirst, $connSip }' f_flows.txt | t2plot
- tawk '{ print $pktsSnt }' f_flows.txt | t2plot -H 10 -sx "0:40"
- t2plot -o pktsSnt f_flows.txt
- t2plot -D -o srcIPCC:l7BytesSnt f_flows.txt
- tawk '{ print proto(), $l7BytesSnt }' f_flows.txt | t2plot -D
- t2plot file_with_one_two_or_three_columns.txt
- t2plot -o "26:28" file_with_many_columns.txt
- t2plot -o "l7BytesSnt:l7BytesRcvd" file_with_many_columns.txt
- Try t2plot -e for more examples.

1.22 t2plugin

Use this script to create a new plugin or list existing plugins For a more comprehensive description of how to write a plugin, refer to Appendix A (Creating a custom plugin) of $T2HOME/doc/documentation.pdf.

1.23 t2rrd

Store Tranalyzer monitoring output into a RRD database (-m option) or use the RRD database generated with -m to monitor and plot various values, e.g., number of flows.

1.24 t2stat

Sends USR1 signal to Tranalyzer to produce intermediary report. The signal sent can be changed with the -SIGNAME option, e.g., t2stat -USR2 or t2stat -INT. If Tranalyzer was started as root, the -s option can be used to run the command with sudo. The -p option can be used to print the PID of running Tranalyzer instances and the -l option provides additional information about the running instances (command and running time). The -i option can be used to cycle through all the running instances and will prompt for confirmation before sending the signal to a specific process. If a numeric argument *N* is provided, sends the signal every *N* seconds, e.g., t2stat 10 to report every 10s. Use t2stat --help for more information.

1.25 t2timeline

Timeline plot of flows: `t2timeline FILE_flows.txt`

- To use relative time, i.e., starting at 0, use the `-r` option.
- The vertical space between A and B flows can be adapted with the `-v` option, e.g., `-v 50`.
- When hovering over a flow, the following information is displayed: `flowInd_flowStat_srcIP:srcPort_dstIP:dstPort_l4Proto_vlanID`.
- Additional information can be displayed with the `-e` option, e.g, `-e macS,macD,duration`
- Use `t2timeline --help` for more information.

An example graph is depicted in Figure 1.

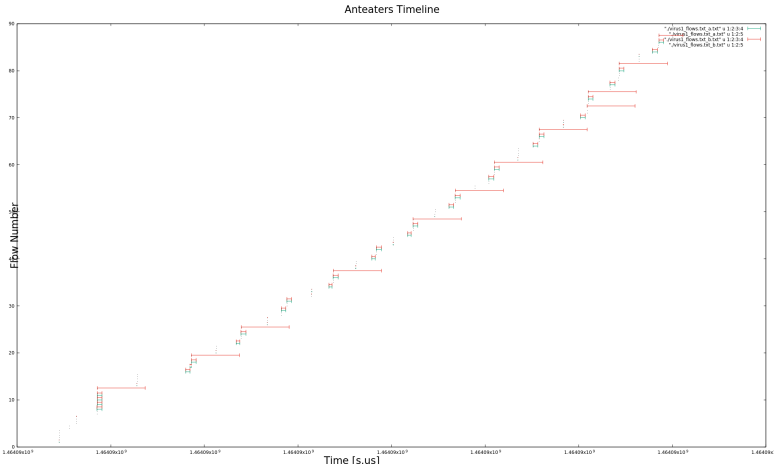

Figure 1: *T2 timeline flow plot*

1.26 t2topcap

Convert pcapng files to pcap: `t2topcap file.pcapng` Try `t2topcap --help` for more information.

1.27 t2utils.sh

Collection of bash functions and variables (`readonly`).

1.27.1 Usage

To access the functions and variables provided by this file, source it in your script as follows:

```
source "$(dirname "${0}")/t2utils.sh"
```

Note that if your script is not in the `scripts/` folder, you will need to adapt the path above to `t2utils.sh` accordingly.

[ZSH] If writing a script for ZSH, add the following line **BEFORE** sourcing the script:

```
unsetopt function_argzero
```

1.27.2 Colors

Alternatives to `printbold`, `printerr`, `printinf`, `printok` and `printwrn`:

Variable	Description	Example
`${BLUE}`	Set the color to blue	`printf "${BLUE}%s${NOCOLOR}\n" "msg"`
`${GREEN}`	Set the color to green	`printf "${GREEN}%s${NOCOLOR}\n" "msg"`
`${ORANGE}`	Set the color to orange	`printf "${ORANGE}%s${NOCOLOR}\n" "msg"`
`${RED}`	Set the color to red	`printf "${RED}%s${NOCOLOR}\n" "msg"`
`${BLUE_BOLD}`	Set the color to blue bold	`printf "${BLUE_BOLD}msg${NOCOLOR}\n"`
`${GREEN_BOLD}`	Set the color to green bold	`printf "${GREEN_BOLD}msg${NOCOLOR}\n"`
`${ORANGE_BOLD}`	Set the color to orange bold	`printf "${ORANGE_BOLD}msg${NOCOLOR}\n"`
`${RED_BOLD}`	Set the color to red bold	`printf "${RED_BOLD}msg${NOCOLOR}\n"`
`${BLUE_ITALIC}`	Set the color to blue italic	`printf "${BLUE_BOLD}${msg}${NOCOLOR}\n"`
`${GREEN_ITALIC}`	Set the color to green italic	`printf "${GREEN_BOLD}${msg}${NOCOLOR}\n"`
`${ORANGE_ITALIC}`	Set the color to orange italic	`printf "${ORANGE_BOLD}${msg}${NOCOLOR}\n"`
`${RED_ITALIC}`	Set the color to red italic	`printf "${RED_BOLD}${msg}${NOCOLOR}\n"`
`${BLUE_UNDERLINE}`	Set the color to blue underline	`echo -e "${BLUE_ITALIC}msg${NOCOLOR}"`
`${GREEN_UNDERLINE}`	Set the color to green underline	`echo -e "${GREEN_ITALIC}msg${NOCOLOR}"`
`${ORANGE_UNDERLINE}`	Set the color to orange underline	`echo -e "${ORANGE_ITALIC}msg${NOCOLOR}"`
`${RED_UNDERLINE}`	Set the color to red underline	`echo -e "${RED_ITALIC}msg${NOCOLOR}"`
`${BOLD}`	Set the font to bold	`echo -e "${BOLD}${msg}${NOCOLOR}"`
`${ITALIC}`	Set the font to italic	`echo -e "${ITALIC}${msg}${NOCOLOR}"`
`${UNDERLINE}`	Set the font to underline	`echo -e "${UNDERLINE}${msg}${NOCOLOR}"`
`${STRIKETHROUGH}`	Set the font to strikethrough	`echo -e "${STRIKETHROUGH}${msg}${NOCOLOR}"`
`${NOCOLOR}`	Reset the color	`echo -e "${BOLD}${msg}${NOCOLOR}"`

1.27.3 Folders

Variable	Description	Example
`${SHOME}`	Points to the folder where the script resides	For `macRecorder/utils/mconv`, `${SHOME}` is `${T2PLHOME}/macRecorder/utils`
`${T2HOME}`	Points to the root folder of Tranalyzer	`cd "${T2HOME}/tranalyzer2"`
`${T2PLHOME}`	Points to the root folder of Tranalyzer plugins	`cd "${T2PLHOME}/t2PSkel"`

1.27.4 Scripts and Programs

Functions and variables pointing to programs.

Functions

Name	Program	Example
AWK	gawk	AWK '{ print }' file
AWKF	gawk -F'\t' -v OFS='\t'	AWKF '{ print }' file
T2	Most recent tranalyzer executable in ${T2HOME}/tranalyzer2/	T2 -r file.pcap
T2B2T	Most recent t2b2t executable in ${T2HOME}/utils/t2b2t/	T2B2T -r file.bin -w file.txt
T2WHOIS	Most recent t2whois executable in ${T2HOME}/utils/t2whois/	T2WHOIS 1.2.3.4

Variables

Name	Program	Example
`${AWK_EXEC}`	`gawk`	`"${AWK_EXEC}" 'NR > 0 { print $2 }' file`
`${OPEN}`	`xdg-open` (Linux), `open` (macOS)	`"${OPEN}" file.pdf`
`${PYTHON}`	`python3`, `python` or `python2`	`"${PYTHON}" file.py`
`${READLINK}`	`readlink` (Linux) / `greadlink` (macOS)	`"${READLINK}" file`
`${SED}`	`sed` (Linux) / `gsed` (macOS)	`"${SED}" 's/ /_/g' << "$str"`
`${T2BUILD}`	`"${T2HOME}/autogen.sh"`	`"${T2BUILD}" tranalyzer2`
`${T2CONF}`	`"${T2HOME}/scripts/t2conf/t2conf"`	`"${T2CONF}" basicFlow -D BFO_MAC=1`
`${T2PLOT}`	`"${T2HOME}/scripts/t2plot"`	`"${T2PLOT}" -o srcIPCC file`
`${TAWK}`	`"${T2HOME}/scripts/tawk/tawk"`	`"${TAWK}" '{ print tuple4() } file'`

1.27.5 Functions

Function	Description
`printbold "msg"`	print a message (bold) with a newline
`printerr "msg"`	print an error message (red) with a newline
`printinf "msg"`	print an info message (blue) with a newline
`printok "msg"`	print an ok message (green) with a newline
`printwrn "msg"`	print a warning message (orange) with a newline
`printfbold "msg"`	print a message (bold) without a newline
`printferr "msg"`	print an error message (red) without a newline
`printfinf "msg"`	print an info message (blue) without a newline
`printfok "msg"`	print an ok message (green) without a newline
`printfwrn "msg"`	print a warning message (orange) without a newline
`fatal "msg"`	print an error message (red) with a newline and exit with status 1
`check_dependency "bin" "pkg"`	check whether a dependency exists (Linux/macOS)
`check_dependency_linux "bin" "pkg"`	check whether a dependency exists (Linux)
`check_dependency_macos "bin" "pkg"`	check whether a dependency exists (macOS)
`test_min_version "ver" "req"`	return true if version number `ver` is greater than or equal to `req`
`has_define "file" "name"`	return 0 if the macro `name` exists in `file`, 1 otherwise
`get_define "name" "file"`	return the value of the macro `name` in `file`
`set_define "name" "value" "file"`	set the value of the macro `name` in `file` to `value`
`replace_suffix "name" "old" "new"`	replace the `old` suffix in `name` by `new`
`join_by "sep" "values..."`	join `values...` with a separator `sep`
`ask_default_no "msg" ["answer"]`	ask the question `msg (y/N)?`, read the answer and assume `no` if answer is not `[yY]` or `[yY][eE][sS]`. Return 0 if answer is

Function	Description
`ask_default_yes "msg" ["answer"]`	`yes`. `answer` can be set to `yes` or `no` to force the answer ask the question `msg (Y/n)?`, read the answer and assume `yes` if answer is not `[nN]` or `[nN][oO]`. Return 0 if answer is `yes`. `answer` can be set to `yes` or `no` to force the answer
`find_most_recent_dir "dir" "dirname"`	recursively find the most recent `dirname` in a directory `dir`
`find_most_recent_file "dir" "filename"`	recursively find the most recent `filename` in a directory `dir`
`t2_build_exec "/path/to/exec" [force]`	ask whether to build the given executable if it does not exist (set `force` to 1 to rebuild `exec` regardless)
`get_t2_exec`	search for `tranalyzer` executable in `${T2HOME}/tranalyzer2` (return an empty string, if it could not be found)
`get_t2b2t_exec`	search for t2b2t executable in `${T2HOME}/utils/t2b2t` (return an empty string, if it could not be found)
`get_t2whois_exec`	search for t2whois executable in `${T2HOME}/utils/t2whois` (return an empty string, if it could not be found)
`abort_if_t2_exec_not_found`	search for `tranalyzer` executable in `${T2HOME}/tranalyzer2`. Abort with status 1 and an error and info message if it could not be found.
`abort_if_t2b2t_exec_not_found`	search for t2b2t executable in `${T2HOME}/utils/t2b2t`. Abort with status 1 and an error and info message if it could not be found.
`abort_if_t2whois_exec_not_found`	search for t2whois executable in `${T2HOME}/utils/t2whois`. Abort with status 1 and an error and info message if it could not be found.
`t2_wget "url" ["outfile"]`	download the data at `url` (set `outfile` (optional) to change the output path)
`t2_wget_n "url" ["outfile"]`	same as `t2_wget`, but turn on timestamping (see `wget -N` option)
`get_nproc`	return the number of processing units available
`validate_float "float"`	return 0 if `float` is a valid floating point value, 1 otherwise
`validate_int "int"`	return 0 if `int` is a valid integer, 1 otherwise
`validate_num "num"`	return 0 if `num` is a valid positive integer, 1 otherwise
`validate_ip "string"`	return 0 if `string` is a valid IPv4 address, 1 otherwise
`validate_pcap "file"`	return 0 if `file` is a valid PCAP file, 1 otherwise
`validate_next_arg "curr" "next"`	check whether `next` exists and is not an option (`curr` is used for error reporting)
`validate_next_arg_exists "curr" "next"`	check whether `next` exists
`validate_next_dir "curr" "next"`	check whether `next` exists and is a directory
`validate_next_file "curr" "next"`	check whether `next` exists and is a regular file

Function	Description
`validate_next_file_or_dir "curr" "next"`	check whether `next` exists and is a regular file or directory
`validate_next_pcap "curr" "next"`	check whether `next` exists and is a PCAP file
`validate_next_num "curr" "next"`	check whether `next` exists and is a positive integer
`validate_next_int "curr" "next"`	check whether `next` exists and is an integer
`validate_next_float "curr" "next"`	check whether `next` exists and is a float
`arg_is_option "arg"`	check whether `arg` exists and is an option (starts with `-`)
`abort_missing_arg "option"`	print an error about a missing argument and exit with status 1
`abort_option_unknown "option"`	print an error about an unknown option and exit with status 1
`abort_required_file`	print an error about a missing required file and exit with status 1
`abort_required_dir`	print an error about a missing required directory and exit with status 1
`abort_required_file_or_dir`	print an error about a missing required file or directory and exit with status 1
`abort_with_help`	print a message explaining how to get help and exit with status 1

1.28 t2viz

Generates a graphviz script which can be loaded into xdot or dotty: `t2viz FILE_flows.txt`.
Accepts T2 flow or packet files with header description.
Try `t2viz --help` for more information.

1.29 t2voipconv

Convert raw audio files extracted with `voipDetector` to wav: `t2voipconv /tmp/TranVoIP`.
Try `t2voipconv --help` for more information.

1.30 t2whois

Query Tranalyzer subnet databases to get geolocation information about IP addresses:

- Get geo info for one IP address: `t2whois 1.2.3.4`
- Get geo info for multiple IPs: `t2whois 1.2.3.4 5.6.7.8`
- Get geo info for one or multiple IPs listed in a file (one IP per line): `t2whois -r file.txt`
- Get geo info for one or multiple IPs entered to stdin via a prompt: `t2whois` or `t2whois -r -`
- Get geo info for one or multiple IPs entered to stdin via a pipe: `cat file.txt | t2whois -q -l`
- Format the output (one line per IP, no header): `t2whois -l -H -r file.txt`
- Format the output (one line per IP, comma separated): `t2whois -l -s "," -r file.txt`

- Query information about the databases `t2whois -V`
- Try `t2whois -h` for more information

1.31 topNStat

Generates sorted lists of all the columns (names or numbers) provided. A list of examples can be displayed using the `-e` option.

2 PDF Report Generation from PCAP using t2fm

2.1 Introduction

This tutorial presents `t2fm`, a script which generates a PDF report out of a PCAP file. Information provided in the report includes top source and destination addresses and ports, protocols and applications, DNS and HTTP activity and potential warnings, such as executable downloads or SSH connections.

2.2 Prerequisites

For this tutorial, it is assumed the user has a basic knowledge of Tranalyzer and that the file `t2_aliases` has been sourced in `~/.bashrc` or `~/.bash_aliases` as follows[2] (make sure to replace `$T2HOME` with the actual path, e.g., `$HOME/tranalyzer2-0.9.3`):

```
# $HOME/.bashrc

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

2.2.1 Required plugins

The following plugins must be loaded for `t2fm` to produce a useful report:

- `basicFlow`
- `basicStats`
- `txtSink`

2.2.2 Optional plugins

The following plugins are optional:

- `arpDecode`
- `dnsDecode`
- `geoip`
- `pwX`
- `sshDecode`
- `sslDecode`
- `httpSniffer`, configured as follows[3]:
  - `HTTP_SAVE_IMAGE=1`
  - `HTTP_SAVE_VIDEO=1`
  - `HTTP_SAVE_AUDIO=1`
  - `HTTP_SAVE_MSG=1`
  - `HTTP_SAVE_TEXT=1`
  - `HTTP_SAVE_APPL=1`
- `nDPI`, configured as follows:
  - `NDPI_OUTPUT_STR=1`
- `portClassifier`, configured as follows:
  - `PBC_NUM=1`
  - `PBC_STR=1`

If one of those plugin is not loaded, messages like `N/A: dnsDecode plugin required` will be displayed in the PDF where the information could not be accessed.

[2] Refer to the file `README.md` or to the documentation for more details
[3] This is only required to report information about EXE downloaded

2.2.3 Packages

The following packages are required to build the PDF:

- texlive-latex-extra
- texlive-fonts-recommended

2.3 PCAP to PDF in One Command

For simplicity, this tutorial assumes the user wants a complete report, i.e., requires all of the optional plugins. The `-b` option builds and configures Tranalyzer and the plugins, the `-A` option opens the generated report.

1. Run `t2fm` directly on the PCAP file (the report will be named `file.pdf`):

```
t2fm -b -A -r file.pcap
```

2.4 Step-by-Step Instructions (PCAP to PDF)

Alternatively if you prefer to configure Tranalyzer, build the plugins and open the generated report yourself:

1. Make sure all the plugins are configured as described in Section 2.2
2. Build Tranalyzer and the plugins [4]:
   `t2build tranalyzer2 basicFlow basicStats txtSink arpDecode dnsDecode geoip \`
   `httpSniffer nDPI portClassifier pwX sshDecode sslDecode`
   (Note that those first two steps can be omitted if `t2fm -b` option is used)
3. Run `t2fm` directly on the PCAP file (the report will be named `file.pdf`):
   `t2fm -r file.pcap`
4. Open the generated PDF report `file.pdf` (Note that this step can be omitted if `t2fm -A` option is used):
   `evince file.pdf`

2.5 Step-by-Step Instructions (flow file to PDF)

Alternatively, if you prefer to run Tranalyzer yourself or already have access to a flow file, replace step 3 of Section 2.4 with the following steps:

1. Run Tranalyzer on a pcap file as follows:
   `t2 -r file.pcap -w out`
2. The previous command should have created the following files:
   `out_headers.txt`
   `out_flows.txt`
3. Run the `t2fm` script on the flow file generated previously:
   `t2fm -F out_flows.txt`

[4] Hint: use the tab completion to avoid typing the full name of all the plugins: `t2build tr<tab> ... ht<tab> ...`

2.6 Step-by-Step Instructions (ClickHouse / MongoDB / PostgreSQL to PDF)

If the `clickhouseSink` , `mongoSink` or `psqlSink` plugins were loaded, t2fm can use the created databases to generate the report (faster).

1. Follow point 1 and 2 from Section 2.4[5]
2. Build the `clickhouseSink` , `mongoSink` or `psqlSink` plugin:
   - **ClickHouse:** `t2build clickhouseSink`
   - **MongoDB:** `t2build mongoSink`
   - **postgreSQL:** `t2build psqlSink`
3. Run Tranalyzer on a pcap file as follows:
   `t2 -r file.pcap -w out`
4. Run the `t2fm` script on the database generated previously:
   - **ClickHouse:** `t2fm -C tranalyzer`
   - **MongoDB:** `t2fm -m tranalyzer`
   - **postgreSQL:** `t2fm -p tranalyzer`

When generating a report from a database a time range to query can be specified with the `-T` option. The complete format is as follows: `YYYY-MM-DD HH:MM:SS.USEC([+-]OFFSET|Z)`, e.g., `2022-08-23 12:34:56.912345+0100`. Note that only the required fields must be specified, e.g., `2022-08-23` is equivalent to `2022-08-23 00:00:00.000000`. For example, to generate a report from the 1st of July to the 11. of August 2022 at 14:59 from a PostgreSQL database, run the following command: `t2fm -p tranalyzer -T "2022-07-01" "2022-08-11 14:59"`

2.7 NetFlow Records to PDF in One Command

For simplicity, this tutorial assumes the user wants a complete report. The `-A` option opens the generated report.

1. Run `t2fm` directly on the NetFlow records (file or directory) (the report will be named `2024.pdf`):

```
t2fm -A -N netflow/2024
```

2.8 Conclusion

This tutorial has presented how `t2fm` can be used to create a PDF report summarizing the traffic contained in a PCAP file or NetFlow records. Although not discussed in this tutorial, it is also possible to use `t2fm` on a live interface (`-i` option) or on a list of PCAP files (`-R` option). For more details, refer to `t2fm` man page or use `t2fm --help`.

[5] `HTTP_SAVE_*` do not need to be set as EXE downloads detection is currently not implemented in the DB backends

3 tawk

3.1 Description

This document describes tawk and its functionalities. tawk works just like awk, but provides access to the columns via their names. In addition, it provides access to helper functions, such as `host()` or `port()`. Custom functions can be added in the folder named `t2custom` where they will be automatically loaded.

3.2 Dependencies

gawk version 4.1 is required.

Ubuntu:	`sudo apt-get install gawk`
Arch:	`sudo pacman -S gawk`
Gentoo:	`sudo emerge gawk`
openSUSE:	`sudo zypper install gawk`
Red Hat/Fedora[6]**:**	`sudo dnf install gawk`
macOS[7]**:**	`brew install gawk`

3.3 Installation

The recommended way to install `tawk` is to install `t2_aliases` as documented in `README.md`:

- Append the following line to `~/.bashrc` (make sure to replace `$T2HOME` with the actual path, e.g., `$HOME/tranalyzer2-0.9.3`):

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

3.3.1 Man Pages

The man pages for `tawk` and `t2nfdump` can be installed by running: `./install.sh man`. Once installed, they can be consulted by running `man tawk` and `man t2nfdump` respectively.

3.4 Usage

- To list the column numbers and names: `tawk -l file_flows.txt`
- To list the column numbers and names as 3 columns: `tawk -l=3 file_flows.txt`
- To list the available functions: `tawk -g file_flows.txt`
- To list the available functions as 3 columns: `tawk -g=3 file_flows.txt`
- To save the original filename and filter used: `tawk -c 'FILTER' file_flows.txt > file.txt`

[6] If the `dnf` command could not be found, try with `yum` instead
[7] Brew is a packet manager for macOS that can be found here: https://brew.sh

- To extract all ICMP flows and the header: `tawk 'hdr() || $l4Proto == 1' file_flows.txt > icmp.txt`
- To extract all ICMP flows without the header: `tawk -H 'icmp()' file_flows.txt > icmp.txt`
- To extract the flow with index 1234: `tawk '$flowInd == 1234' file_flows.txt`
- To extract all DNS flows and the header: `tawk 'hdr() || strtonum($dnsStat)' file_flows.txt`
- To consult the documentation for the function 'func': `tawk -d func`
- To consult the documentation for the functions 'min' and 'max': `tawk -d min,max`
- To consult the documentation for all the available functions: `tawk -d all`
- To consult the documentation for the variable 'var': `tawk -V var`
- To consult the documentation for the variable 'var' with value `0x8a`: `tawk -V var=0x8a`
- To decode all variables from `tranalyzer2` log file: `tawk -L out_log.txt`
- To decode all variables from `tranalyzer2` log file (stdout): `t2 -r file.pcap | tawk -L`
- To convert the output to JSON: `tawk 'json($flowStat "\t" tuple5())' file_flows.txt`
- To convert the output to JSON: `tawk 'aggr(tuple2())' file_flows.txt | tawk 'json()'`
- To create a PCAP with all packets from flow 5: `tawk -x flow5.pcap '$flowInd == 5' file_flows.txt`
- To create a PCAP with packets 4-9: `tawk -P -x pkts-4_to_9.pcap 'packet("4-9")' file_packets.txt`
- To see all ICMP packets in Wireshark: `tawk -k 'icmp()' file_flows.txt`
- To see packet 4, 10 and 42 in Wireshark: `tawk -P -k 'packet("4;10;42")' file_packets.txt`

For a complete list of options, use the `-h` option.

Note that an option not recognized by tawk is internally passed to awk/gawk. One of the most useful is the `-v` option to set the value of a variable:

- Changing the output field separator:
  `tawk -v OFS=',' '{ print $col1, $col2 }' file.txt`
- Passing a variable to tawk:
  `tawk -v myvar=myvalue '{ print $col1, myvar }' file.txt`

For a complete list of options, run `awk -h`.

3.5 -s and -N Options

The `-s` option can be used to specify the starting character(s) of the row containing the column names (default: `'%'`). If several rows start with the specified character(s), then the last one is used as column names. To change this behavior, the line number can be specified as well with the help of the `-N` option. For example, if rows 1 to 5 start with `'#'` and row 3 contains the column names, specify the separator as follows: `tawk -s '#' -N 3` If the row with column names does not start with a special character, use `-s ''`.

3.6 Related Utilities

3.6.1 awkf

Configure `awk` to use tabs, i.e., '`\t`' as input and output separator (prevent issue with repetitive values), e.g., `awkf '{ print $4 }' file_flows.txt`

3.6.2 lsx

Display columns with fixed width (default: 40), e.g., `lsx file_flows.txt` or `lsx 45 file_flows.txt`

3.6.3 sortu

Sort rows and count the number of times a given row appears, then sort by the most occurring rows. (Alias for `sort | uniq -c | sort -rn`). Useful, e.g., to analyze the most occurring user-agents: `tawk '{ print $httpUsrAg }' FILE_flows.txt | sortu`

sortup
Same as `sortu`, but display the relative percentage instead of the absolute count. e.g., to analyze the most occurring user-agents: `tawk '{ print $httpUsrAg }' FILE_flows.txt | sortup`

3.6.4 tcol

Display columns with minimum width, e.g., `tcol file_flows.txt`.

3.7 Functions

Collection of functions for `tawk`:

- Parameters between brackets are optional,
- IPs can be given as string (`"1.2.3.4"`), hexadecimal (`0xffffffff`) or int (`4294967295`),
- Network masks can be given as string (`"255.255.255.0"`), hexadecimal (`0xffffff00`) or CIDR notation (`24`),
- Networks can be given as string, hexadecimal or int, e.g., `"1.2.3.4/24"` or `"0x01020304/255.255.255.0"`,
- String functions can be made case insensitive by adding the suffix `i`, e.g., `streq` → `streqi`,
- Some examples are provided below,
- More details and examples can be found for every function by running `tawk -d funcname`.

Function	Description
`hdr()`	Use this function in your tests to keep the header (column names).
`tuple2()`	Return the 2 tuple (source IP and destination IP).
`tuple3()`	Return the 3 tuple (source IP, destination IP and port).
`tuple4()`	Return the 4 tuple (source IP and port, destination IP and port).
`tuple5()`	Return the 5 tuple (source IP and port, destination IP and port, protocol).

Function	Description
`tuple6()`	Return the 6 tuple (source IP and port, dest. IP and port, proto, VLANID).
`host([ip|net])`	Return true if the source or destination IP is equal to `ip` or belongs to `net`. If `ip` is omitted, return the source and destination IP.
`shost([ip|net])`	Return true if the source IP is equal to `ip` or belongs to `net`. If `ip` is omitted, return the source IP.
`dhost([ip|net])`	Return true if the destination IP is equal to `ip` or belongs to `net`. If `ip` is omitted, return the destination IP.
`net([ip|net])`	Alias for `host([ip|net])`.
`snet([ip|net])`	Alias for `shost([ip|net])`.
`dnet([ip|net])`	Alias for `dhost([ip|net])`.
`loopback(ip)`	Return true if `ip` is a loopback address.
`mcast(ip)`	Return true if `ip` is a multicast address.
`privip(ip)`	Return true if `ip` is a private IP.
`port([p])`	Return true if the source or destination port appears in `p` (comma or semicolon separated). Ranges may also be specified using a dash, e.g., `port("1-3")`. If `p` is omitted, return the source and destination port.
`dport([p])`	Return true if the destination port appears in `p` (comma or semicolon separated). Ranges may also be specified using a dash, e.g., `dport("1-3")`. If `p` is omitted, return the destination port.
`sport([p])`	Return true if the source port appears in `p` (comma or semicolon separated). Ranges may also be specified using a dash, e.g., `sport("1-3")`. If `p` is omitted, return the source port.
`ip()`	Return true if the flow contains IPv4 or IPv6 traffic.
`ipv4()`	Return true if the flow contains IPv4 traffic.
`ipv6()`	Return true if the flow contains IPv6 traffic.
`proto([p])`	Return true if the protocol number appears in `p` (comma or semicolon separated). Ranges may also be specified using a dash, e.g., `proto("1-3")`. If `p` is omitted, return the protocol number.
`proto2str([p])`	Return the string representation of the protocol number `p`. If `p` is omitted, return the string representation of the protocol.
`icmp([p])`	Return true if the protocol is equal to 1 (ICMP).
`igmp([p])`	Return true if the protocol is equal to 2 (IGMP).
`tcp([p])`	Return true if the protocol is equal to 6 (TCP).
`udp([p])`	Return true if the protocol is equal to 17 (UDP).
`rsvp([p])`	Return true if the protocol is equal to 46 (RSVP).
`gre([p])`	Return true if the protocol is equal to 47 (GRE).
`esp([p])`	Return true if the protocol is equal to 50 (ESP).
`ah([p])`	Return true if the protocol is equal to 51 (AH).
`icmp6([p])`	Return true if the protocol is equal to 58 (ICMPv6).

Function	Description
sctp([p])	Return true if the protocol is equal to 132 (SCTP).
dhcp()	Return true if the flow contains DHCP traffic.
dns()	Return true if the flow contains DNS traffic.
http()	Return true if the flow contains HTTP traffic.
tcpflags([val])	If val is specified, return true if the specified flags are set. If val is omitted, return a string representation of the TCP flags.
ip2num(ip)	Convert an IP address to a number.
ip2hex(ip)	Convert an IPv4 address to hex.
ip2str(ip)	Convert an IPv4 address to string.
ip62str(ip)	Convert an IPv6 address to string.
ip6compress(ip)	Compress an IPv6 address.
ip6expand(ip[,trim])	Expand an IPv6 address. If trim is different from 0, remove leading zeros.
ip2mask(ip)	Convert an IP address to a network mask (int).
mask2ip(m)	Convert a network mask (int) to an IPv4 address (int).
mask2ipstr(m)	Convert a network mask (int) to an IPv4 address (string).
mask2ip6(m)	Convert a network mask (int) to an IPv6 address (int).
mask2ip6str(m)	Convert a network mask (int) to an IPv6 address (string).
ipinnet(ip,net[,mask])	Test whether an IP address belongs to a given network.
ipinrange(ip,low,high)	Test whether an IP address lies between two addresses.
localtime(t)	Convert UNIX timestamp to string (localtime).
utc(t)	Convert UNIX timestamp to string (UTC).
timestamp(t)	Convert date to UNIX timestamp.
t2split(val,sep [,num[,osep]])	Split values according to sep. If num is omitted or 0, val is split into osep separated columns. If num > 0, return the num repetition. If num < 0, return the num repetition from the end, e.g., -1 for last element. Multiple num can be specified, e.g., "1;-1;2". Output separator osep, defaults to OFS.
splitc(val[,num[,osep]])	Split compound values. Alias for t2split(val, "_", num, osep).
splitr(val[,num[,osep]])	Split repetitive values. Alias for t2split(val, ";", num, osep).
valcontains(val,sep,item)	Return true if one item of val split by sep is equal to item.
cvalcontains(val,item)	Alias for valcontains(val, "_", item).
rvalcontains(val,item)	Alias for valcontains(val, ";", item).
strisempty(val)	Return true if val is an empty string.
streq(val1,val2)	Return true if val1 is equal to val2.

Function	Description
strneq(val1,val2)	Return true if val1 and val2 are not equal.
hasprefix(val,pre)	Return true if val begins with the prefix pre.
hassuffix(val,suf)	Return true if val finished with the suffix suf.
contains(val,txt)	Return true if val contains the substring txt.
not(q)	Return the logical negation of a query q. This function must be used to keep the header when negating a query.
bfeq(val1,val2)	Return true if the hexadecimal numbers val1 and val2 are equal.
bitsallset(val,mask)	Return true if all the bits set in mask are also set in val.
bitsanyset(val,mask)	Return true if one of the bits set in mask is also set in val.
isset(v)	Return true if v is set, i.e., not empty, false otherwise.
isfloat(v)	Return true if v is a floating point number.
isint(v)	Return true if v is an integer.
isnum(v)	Return true if v is a number (signed, unsigned or floating point).
isuint(v)	Return true if v is an unsigned integer.
isip(v)	Return true if v is an IPv4 address in hexadecimal, numerical or dotted decimal notation.
isip6(v)	Return true if v is an IPv6 address.
isiphex(v)	Return true if v is an IPv4 address in hexadecimal notation.
isipnum(v)	Return true if v is an IPv4 address in numerical (int) notation.
isipstr(v)	Return true if v is an IPv4 address in dotted decimal notation.
join(a,s)	Convert an array to string, separating each value with s.
quote(s)	Add leading and trailing quotes to a string s and escape all quotes in s.
unquote(s)	Remove leading and trailing quotes from a string s and unespace all escaped quotes in s.
chomp(s)	Remove leading and trailing spaces from a string s.
strip(s)	Remove leading and trailing spaces from a string s.
lstrip(s)	Remove leading spaces from a string s.
rstrip(s)	Remove trailing spaces from a string s.
ientropy([num[,sc[,rev[,imin]]]])	Compute the Shannon (information) entropy of each column. Set imin to filter out columns with low entropy ($\leq$ *imin*).
mean(c)	Compute the mean value of a column c. The result can be accessed with get_mean(c) or printed with print_mean([c]).
min(c)	Keep track of the min value of a column c. The result can be accessed with get_min(c) or printed with print_min([c]).
max(c)	Keep track of the max value of a column c. The result can be accessed with get_max(c) or

Function	Description
	printed with `print_max([c])`.
`abs(v)`	Return the absolute value of `v`.
`log2(n)`	Return the binary logarithm (log base 2) of `a`.
`max2(a,b)`	Return the maximum value between `a` and `b`.
`max3(a,b,c)`	Return the maximum value between `a`, `b` and `c`.
`min2(a,b)`	Return the minimum value between `a` and `b`.
`min3(a,b,c)`	Return the minimum value between `a`, `b` and `c`.
`aggr(fields[,val[,num]])`	Perform aggregation of `fields` and store the sum of `val`. `fields` and `val` can be tab separated lists of fields, e.g., `$srcIP"\t"$dstIP`. Results are sorted according to the first value of `val`. If `val` is omitted, the empty string or equal to `"flows"` or `"packets"` (case insensitive), count the number of records (flows or packets). If `num` is omitted or 0, return the full list. If `num > 0` return the top `num` results. If `num < 0` return the bottom `num` results.
`aggrrep(fields[,val[,num[,ign_e[,sep]]]])`	Perform aggregation of the repetitive `fields` and store the sum of `val`. `val` can be a tab separated lists of fields, e.g., `$l7BytesSnt"\t"$pktsSnt`. Results are sorted according to the first value of `val`. If `val` is omitted, the empty string or equal to `"flows"` or `"packets"` (case insensitive), count the number of records (flows or packets). If `num` is omitted or 0, return the full list. If `num > 0` return the top `num` results. If `num < 0` return the bottom `num` results. If `ign_e` is omitted or 0, consider all values, otherwise ignore empty values. `sep` can be used to change the separator character (default: `";"`).
`t2rsort(col[,num[,type]])`	Sort the file in reverse order according to `col`. (Multiple column numbers can be specified by using `";"` as separator, e.g., `1 ";" 2`) If `num` is omitted or 0, return the full list. If `num > 0` return the top `num` results. If `num < 0` return the bottom `num` results. `type` can be used to specify the type of data to sort: `"ip"`, `"num"` or `"str"` (default is based on the first matching record).
`t2sort(col[,num[,type[,rev]]])`	Sort the file according to `col`. (Multiple column numbers can be specified by using `";"` as separator, e.g., `1 ";" 2`) If `num` is omitted or 0, return the full list. If `num > 0` return the top `num` results. If `num < 0` return the bottom `num` results. `type` can be used to specify the type of data to sort:

Function	Description
	"ip", "num" or "str" (default is based on the first matching record). If rev > 0, sort in reverse order (alternatively, use the t2rsort() function).
t2whois(ip[,o_opt])	Wrapper to call t2whois from tawk. ip must be a valid IPv4/6 address. o_opt is passed verbatim to t2whois -o option (run t2whois -L for more details).
wildcard(expr)	Print all columns whose name matches the regular expression expr. If expr is preceded by an exclamation mark, return all columns whose name does **NOT** match expr.
hrnum(num[,mode[,suffix]])	Convert the number num to human readable form.
hrtime(secs[,mode[,unit]])	Convert the timestamp (seconds) secs to human readable form.
json([s])	Convert the string s to JSON. The first record is used as column names. If s is omitted, convert the entire row.
texscape(s)	Escape the string s to make it LaTeX compatible.
bitshift(n,t[,d[,b]])	Shift a byte or of a list of bytes n to the left or right by a given number of bits t. To shift to the left, set d to 0 (default), to shift to the right set d $\neq$ 0. Set b to 16 to force interpretation as hexadecimal, e.g., interpret 45 as 69 (0x45) instead of 45.
nibble_swap(n[,b])	Swap the nibbles of a byte or of a list of bytes n. Set b to 16 to force interpretation as hexadecimal, e.g., interpret 45 as 69 (0x45) instead of 45.
tobits(u,[b])	Convert the unsigned integer u to its binary representation. Set b to 16 to force interpretation as hexadecimal, e.g., interpret 45 as 69 (0x45) instead of 45.
base64(s)	Encode a string s as base64.
base64d(s)	Decode a base64 encoded string s.
urldecode(url)	Decode the encoded URL url.
printbold(s, n)	Print the string s in bold with an added newline. If n is set, the trailing newline is omitted.
printerr(s, n)	Print the string s in red to stderr. If n is set, the trailing newline is omitted.
printinf(s, n)	Print the string s in blue. If n is set, the trailing newline is omitted.
printok(s, n)	Print the string s in green. If n is set, the trailing newline is omitted.
printwrn(s, n)	Print the string s in orange. If n is set, the trailing newline is omitted.
diff(file[,mode])	Compare two files (file and the input), and print the name and number of the columns which differ. The mode parameter can be used to control the

Function	Description
	format of the output.
`ffsplit([s[,k[,h]]])`	Split the input file into smaller more manageable files. The files to create can be specified as argument to the function (one comma separated string). If no argument is specified, create one file per column whose name ends with `Stat`, e.g., `dnsStat`, and one for `pwxType` (`pw`) and `covertChannels` (`cc`). If `k` > 0, then only print relevant fields and those controlled by `h`, a comma separated list of fields to keep in each file, e.g., `"srcIP,dstIP"`.
`flow([f])`	Return all flows whose index appears in `f` (comma or semicolon separated). Ranges may also be specified using a dash, e.g., `flow("1-3")`. If `f` is omitted, return the flow index.
`packet([p])`	Return all packets whose number appears in `p` (comma or semicolon separated). Ranges may also be specified using a dash, e.g., `packet("1-3")`. If `p` is omitted, return the packet number.
`follow_stream(f[,of[,d[,pf[,r[,nc]]]]])`	Return the payload of the flow with index `f`. `of` can be used to change the output format [default: 0]: 0: Payload only, 1: prefix each payload with packet/flow info, 2: JSON, 3: Reconstruct (pipe the output to `xxd -p -r` to reproduce the binary file). `d` can be used to only extract a specific direction (`"A"` or `"B"`) [default: `""` (A and B)]. `pf` can be used to change the payload format [default: 0]: 0: ASCII, 1: Hexdump, 2: Raw/Binary, 3: Base64. `r` can be used to prevent the analysis of TCP sequence numbers (no TCP reassembly and reordering). `nc` can be used to print the data without colors.
`shark(q)`	Query flow files according to Wireshark's syntax.

3.8 Examples

Collection of examples using `tawk` functions:

Function	Description
`covertChans([val[,num]])`	Return information about hosts possibly involved in a covert channels. If `val` is omitted or equal to `"flows"`, count the number of flows. Otherwise, sum up the values of `val`.

Function	Description
	If num is omitted or 0, return the full list, If num > 0 return the top num results, If num < 0 return the bottom num results.
dnsZT()	Return all flows where a DNS zone transfer was performed.
exeDL([n])	Return the top N EXE downloads.
httpHostsURL([f])	Return all HTTP hosts and a list of the files hosted (sorted alphabetically). If f > 0, print the number of times a URL was requested.
nonstdports()	Return all flows running protocols over non-standard ports.
passivedns()	Extract all DNS server replies from a flow file. The following information is reported for each reply: FirstSeen, LastSeen, Type (A or AAAA), TTL, Query, Answer, Organization, Country, AS number
passwords([val[,num]])	Return information about hosts sending authentication in cleartext. If val is omitted or equal to "flows", count the number of flows. Otherwise, sum up the values of val. If num is omitted or 0, returns the full list. If num > 0 return the top num results. If num < 0 return the bottom num results.
postQryStr([n])	Return the top N POST requests with query strings.
ssh()	Return the SSH connections.
topDnsA([n])	Return the top N DNS answers.
topDnsIp4([n])	Return the top N DNS answers IPv4 addresses.
topDnsIp6([n])	Return the top N DNS answers IPv6 addresses.
topDnsQ([n])	Return the top N DNS queries.
topHttpMimesST([n])	Return the top HTTP content-type (type/subtype).
topHttpMimesT([n])	Return the top HTTP content-type (type only).
topSLD([n])	Return the top N second-level domains queried (google.com, yahoo.com, ...).
topTLD([n])	Return the top N top-level domains (TLD) queried (.com, .net, ...).

3.9 t2nfdump

Collection of functions for tawk allowing access to specific fields using a syntax similar as nfdump.

Function	Description
ts()	Start Time — first seen
te()	End Time — last seen
td()	Duration
pr()	Protocol
sa()	Source Address
da()	Destination Address
sap()	Source Address:Port
dap()	Destination Address:Port
sp()	Source Port
dp()	Destination Port
pkt()	Packets — default input
ipkt()	Input Packets
opkt()	Output Packets
byt()	Bytes — default input
ibyt()	Input Bytes
obyt()	Output Bytes
flg()	TCP Flags
mpls1()	MPLS label 1
mpls2()	MPLS label 2
mpls3()	MPLS label 3
mpls4()	MPLS label 4
mpls5()	MPLS label 5
mpls6()	MPLS label 6
mpls7()	MPLS label 7
mpls8()	MPLS label 8
mpls9()	MPLS label 9
mpls10()	MPLS label 10
mpls()	MPLS labels 1–10
bps()	Bits per second
pps()	Packets per second
bpp()	Bytes per packet
oline()	nfdump line output format (-o line)
olong()	nfdump long output format (-o long)
oextended()	nfdump extended output format (-o extended)

3.10 t2custom

Copy your own functions in this folder. Refer to Section 3.11 for more details on how to write a tawk function. To have your functions automatically loaded, include them in the file t2custom/t2custom.load.

3.11 Writing a tawk Function

- Ideally one function per file (where the filename is the name of the function)
- Private functions are prefixed with an underscore

- Always declare local variables 8 spaces after the function arguments
- Local variables are prefixed with an underscore
- Use uppercase letters and two leading and two trailing underscores for global variables
- Include all referenced functions
- Files should be structured as follows:

```
#!/usr/bin/env awk
#
# Function description
#
# Parameters:
#   - arg1: description
#   - arg2: description (optional)
#
# Dependencies:
#   - plugin1
#   - plugin2 (optional)
#
# Examples:
#   - tawk 'funcname()' file.txt
#   - tawk '{ print funcname() }' file.txt

@include "hdr"
@include "_validate_col"

function funcname(arg1, arg2, [8 spaces] _locvar1, _locvar2) {
    _locvar1 = _validate_col("colname1;altcolname1", _my_colname1)
    _validate_col("colname2")

    if (hdr()) {
        if (__PRIHDR__) print "header"
    } else {
        print "something", $_locvar1, $colname2
    }
}
```

3.12 Using tawk Within Scripts

To use `tawk` from within a script:

1. Create a `TAWK` variable pointing to the script: `TAWK="$T2HOME/scripts/tawk/tawk"`
2. Call `tawk` as follows: `$TAWK 'dport(80)' file.txt`

3.13 Using tawk With Non-Tranalyzer Files

`tawk` can also be used with files which were not produced by Tranalyzer.

- The input field separator can be specified with the `-F` option, e.g., `tawk -F ',' 'program' file.csv`
- The row listing the column names, can start with any character specified with the `-s` option, e.g., `tawk -s '#' 'program' file.txt`
- All the column names must not be equal to a function or builtin name
- Valid column names must start with a letter (`a-z`, `A-Z`) and can be followed by any number of alphanumeric characters or underscores
- If the column names are different from those used by Tranalyzer, refer to Section 3.13.1.

3.13.1 Mapping External Column Names to Tranalyzer Column Names

If the column names are different from those used by Tranalyzer, a mapping between the different names can be made in the file `my_vars`. The format of the file is as follows:

```
BEGIN {
    _my_srcIP = non_t2_name_for_srcIP
    _my_dstIP = non_t2_name_for_dstIP
    ...
}
```

Once edited, run tawk with the `-i $T2HOME/scripts/tawk/my_vars` option and the external column names will be automatically used by tawk functions, such as `tuple2()`. For more details, refer to the `my_vars` file.

3.13.2 Using tawk with Bro/Zeek Files

To use tawk with Bro/Zeek log files, use one of `--bro` or `--zeek` option:

```
tawk -bro '{ program }' file.log tawk -zeek '{ program }' file.log
```

3.14 Awk Cheat Sheet

- Tranalyzer flow files default field separator is '`\t`':
  - **Always** use `awk -F'\t'` (or `awkf`/`tawk`) when working with flow files.
- Load libraries, e.g., tawk functions, with `-i`: `awk -i file.awk 'program' file.txt`
- Always use `strtonum` with hex numbers (bitfields)
- Awk indices start at 1
- Using tawk is recommended.

3.14.1 Useful Variables

- `$0`: entire line
- `$1`, `$2`, ..., `$NF`: column 1, 2, ...
- `FS`: field separator
- `OFS`: output field separator
- `ORS`: output record separator
- `NF`: number of fields (columns)
- `NR`: record (line) number
- `FNR`: record (line) number relative to the current file
- `FILENAME`: name of current file
- To use external variables, use the `-v` option, e.g., `awk -v name="value" '{ print name }' file.txt`.

3.14.2 Awk Program Structure

```
awk -F'\t' -i min -v OFS='\t' -v h="$(hostname)" '
    BEGIN { a = 0; b = 0; }         # Called once at the beginning
    /^A/  { a++ }                   # Called for every row starting with char A
    /^B/  { b++ }                   # Called for every row starting with char B
          { c++ }                   # Called for every row
    END   { print h, min(a, b), c } # Called once at the end
' file.txt
```

3.15 Awk Templates

- Print the whole line:
  - `tawk '{ print }' file.txt`
  - `tawk '{ print $0 }' file.txt`
  - `tawk 'FILTER' file.txt`
  - `tawk 'FILTER { print }' file.txt`
  - `tawk 'FILTER { print $0 }' file.txt`
- Print selected columns only:
  - `tawk '{ print $srcIP, $dstIP }' file.txt`
  - `tawk '{ print $1, $2 }' file.txt`
  - `tawk '{ print $4 "\t" $6 }' file.txt`

```
  – tawk '{
        for (i = 6; i < NF; i++) {
            printf "%s\t", $i
        }
        printf "%s\n", $NF
    }' file.txt
```

- Keep the column names:
  - `tawk 'hdr() || FILTER' file.txt`
  - `awkf 'NR == 1 || FILTER' file.txt`
  - `awkf '/^%/ || FILTER' file.txt`
  - `awkf '/^%[[:space:]]*[[:alpha:]][[:alnum:]_]*$/ || FILTER' file.txt`
- Skip the column names:
  - `tawk '!hdr() && FILTER' file.txt`
  - `awkf 'NR > 1 && FILTER' file.txt`
  - `awkf '!/^%/ && FILTER' file.txt`
  - `awkf '!/^%[[:space:]]*[[:alpha:]][[:alnum:]_]*$/ && FILTER' file.txt`
- Bitfields and hexadecimal numbers:
  - `tawk 'bfeq($3,0)' file.txt`
  - `awkf 'strtonum($3) == 0' file.txt`
  - `tawk 'bitsanyset($3,1)' file.txt`
  - `tawk 'bitsallset($3,0x81)' file.txt`
  - `awkf 'and(strtonum($3), 0x1)' file.txt`
- Split compound values:
  - `tawk '{ print splitc($16, 1) }' file.txt # first element`
  - `tawk '{ print splitc($16, -1) }' file.txt # last element`
  - `awkf '{ split($16, A, "_"); print A[1] }' file.txt`
  - `awkf '{ n = split($16, A, "_"); print A[n] }' file.txt # last element`
  - `tawk '{ print splitc($16) }' file.txt`
  - `awkf '{ split($16, A, "_"); for (i=1;i<=length(A);i++) print A[i] }' file.txt`
- Split repetitive values:
  - `tawk '{ print splitr($16, 3) }' file.txt # third repetition`
  - `tawk '{ print splitr($16, -2) }' file.txt # second to last repetition`
  - `awkf '{ split($16, A, ";"); print A[3] }' file.txt`
  - `awkf '{ n = split($16, A, ";"); print A[n] }' file.txt # last repetition`
  - `tawk '{ print splitr($16) }' file.txt`

  - `awkf '{ split($16, A, ";"); for (i=1;i<=length(A);i++) print A[i] }' file.txt`

- Filter out empty strings:
  - `tawk '!strisempty($4)' file.txt`
  - `awkf '!(length($4) == 0 || $4 == "\"\"")' file.txt`
- Compare strings (case sensitive):
  - `tawk 'streq($3,$4)' file.txt`
  - `awkf '$3 == $4' file.txt`
  - `awkf '$3 == \"text\"' file.txt`
- Compare strings (case insensitive):
  - `tawk 'streqi($3,$4)' file.txt`
  - `awkf 'tolower($3) == tolower($4)' file.txt`
- Use regular expressions on specific columns:
  - `awkf '$8 ~ /^192.168.1.[0-9]{1,3}$/' file.txt # print matching rows`
  - `awkf '$8 !~ /^192.168.1.[0-9]{1,3}$/' file.txt # print non-matching rows`
- Use column names in awk:
  - `tawk '{ print $srcIP, $dstIP }' file.txt`
  - ```
    awkf '
        NR == 1 {
            for (i = 1; i <= NF; i++) {
                if ($i == "srcIP") srcIP = i
                else if ($i == "dstIP") dstIP = i
            }
            if (srcIP == 0 || dstIP == 0) {
                print "No column with name srcIP and/or dstIP"
                exit
            }
        }
        NR > 1 {
            print $srcIP, $dstIP
        }
    ' file.txt
    ```
  - ```
    awkf '
        NR == 1 {
            for (i = 1; i <= NF; i++) {
                col[$i] = i
            }
        }
        NR > 1 {
            print $col["srcIP"], $col["dstIP"];
        }

    ' file.txt
    ```

3.16 Examples

1. Pivoting (variant 1):
   (a) First extract an attribute of interest, e.g., an unresolved IP address in the `Host:` field of the HTTP header:

   ```
   tawk 'aggr($httpHosts)' FILE_flows.txt | tawk '{ print unquote($1); exit }'
   ```

   (b) Then, put the result of the last command in the `badguy` variable and use it to extract flows involving this IP:

   ```
   tawk -v badguy="$(!!)" 'host(badguy)' FILE_flows.txt
   ```

2. Pivoting (variant 2):
   (a) First extract an attribute of interest, e.g., an unresolved IP address in the `Host:` field of the HTTP header, and store it into a `badip` variable:

   ```
   badip="$(tawk 'aggr($httpHosts)' FILE_flows.txt | tawk '{ print unquote($1);exit }')"
   ```

   (b) Then, use the `badip` variable to extract flows involving this IP:

   ```
   tawk -v badguy="$badip" 'host(badguy)' FILE_flows.txt
   ```

3. Aggregate the number of bytes sent between source and destination addresses (independent of the protocol and port) and output the top 10 results:

   ```
   tawk 'aggr($srcIP "\t" $dstIP, $l7BytesSnt, 10)' FILE_flows.txt

   tawk 'aggr(tuple2(), $l7BytesSnt "\t" "Flows", 10)' FILE_flows.txt
   ```

4. Sort the flow file according to the duration (longest flows first) and output the top 5 results:

   ```
   tawk 't2sort(duration, 5)' FILE_flows.txt
   ```

5. Extract all TCP flows while keeping the header (column names):

   ```
   tawk 'hdr() || tcp()' FILE_flows.txt
   ```

6. Extract all flows whose destination port is between 6000 and 6008 (included):

   ```
   tawk 'dport("6000-6008")' FILE_flows.txt
   ```

7. Extract all flows whose destination port is 53, 80 or 8080:

   ```
   tawk 'dport("53;80;8080")' FILE_flows.txt
   ```

8. Extract all flows whose source IP is in subnet 192.168.1.0/24 (using `host` or `net`):

   ```
   tawk 'shost("192.168.1.0/24")' FILE_flows.txt

   tawk 'snet("192.168.1.0/24")' FILE_flows.txt
   ```

9. Extract all flows whose source IP is in subnet 192.168.1.0/24 (using `ipinrange`):

```
tawk 'ipinrange($srcIP, "192.168.1.0", "192.168.1.255")' FILE_flows.txt
```

10. Extract all flows whose source IP is in subnet 192.168.1.0/24 (using `ipinnet`):

```
tawk 'ipinnet($srcIP, "192.168.1.0", "255.255.255.0")' FILE_flows.txt
```

11. Extract all flows whose source IP is in subnet 192.168.1.0/24 (using `ipinnet` and a hex mask):

```
tawk 'ipinnet($srcIP, "192.168.1.0", 0xffffff00)' FILE_flows.txt
```

12. Extract all flows whose source IP is in subnet 192.168.1.0/24 (using `ipinnet` and the CIDR notation):

```
tawk 'ipinnet($srcIP, "192.168.1.0/24")' FILE_flows.txt
```

13. Extract all flows whose source IP is in subnet 192.168.1.0/24 (using `ipinnet` and a CIDR mask):

```
tawk 'ipinnet($srcIP, "192.168.1.0", 24)' FILE_flows.txt
```

For more examples, refer to tawk `-d` option, e.g., `tawk -d aggr`, where every function is documented and comes with a set of examples. The complete documentation can be consulted by running `tawk -d all`.

3.17 FAQ

3.17.1 Can I use tawk with non Tranalyzer files?

Yes, refer to Section 3.13.

3.17.2 Can I use tawk functions with non Tranalyzer column names?

Yes, edit the `my_vars` file and load it using `-i $T2HOME/scripts/tawk/my_vars` option. Refer to Section 3.13.1 for more details.

3.17.3 Can I use tawk with files without column names?

Yes, but you won't be able to use the functions which require a specific column, e.g., `host()`.

3.17.4 The row listing the column names start with a `'#'` instead of a `'%'`... Can I still use tawk?

Yes, use the `-s` option to specify the first character, e.g., `tawk -s '#' 'program'`

3.17.5 Can I process Bro/Zeek log files with tawk?

Yes, use the `--zeek` option.

3.17.6 Can I process a CSV (Comma Separated Value) file with tawk?

The simplest way to process CSV files is to use the `--csv` option. This sets the input and output separators to a comma and considers the first row to be the column names.

```
tawk --csv 'program' file.csv
```

Alternatively, the input field separator can be changed with the `-F` option and the output separator with `-O ','` or `-v OFS=','`. Note that tawk expects the column names to be the last row starting with a `'%'`. This can be changed with the `-s` and `-N` options (Section 3.5).

```
tawk -F ',' -v OFS=',' -s "" -N 1 'program' file.csv
```

3.17.7 Can I produce a CSV (Comma Separated Value) file from tawk?

The output field separator (`OFS`) can be changed with the `-O 'fs'` or `-v OFS='fs'` option. To produce a CSV file, run tawk as follows: `tawk -O ',' 'program' file.txt` or `tawk -v OFS=',' 'program' file.txt`

3.17.8 Can I write my tawk programs in a file instead of the command line?

Yes, copy the program (without the single quotes) in a file, e.g., `prog.txt` and run it as follows:
`tawk -f prog.txt file.txt`

3.17.9 Can I still use column names if I pipe data into tawk?

Yes, you can specify a file containing the column names with the `-I` option as follows:
`cat file.txt | tawk -I colnames.txt 'program'`

3.17.10 Can I use tawk if the row with the column names does not start with a special character?

Yes, you can specify the empty character with `-s ""`. Refer to Section 3.5 for more details.

3.17.11 I get a list of syntax errors from gawk... What is the problem?

The name of the columns is used to create variable names. If it contains forbidden characters, then an error similar to the following is reported.

```
gawk: /tmp/fileBndhdf:3: col-name = 3
gawk: /tmp/fileBndhdf:3:          ^ syntax error
```

Although tawk will try to replace forbidden characters with underscore, the best practice is to use only alphanumeric characters (`A-Z`, `a-z`, `0-9`) and underscore as column names. Note that a column name **MUST NOT** start with a number.

3.17.12 I get a function name previously defined error from gawk... What is the problem?

The name of the columns is used to create variable names. If a column is named after a tawk function or a builtin, then an error similar to the following is reported.

```
gawk: In file included from ah:21,
gawk:                  from /home/user/tranalyzer2/scripts/tawk/funcs/funcs.load:8,
gawk: proto:36: error: function name 'proto' previously defined
```

In this case, you have two options. Either rename the column(s) in your file, e.g., `proto` → `l4Proto` or use `tawk -t` option. With the `-t` option, Tawk tries to validate the column names by ensuring that no column names is equal to a function name and that all column names used in the program exist. Note that this verification process can be slow.

3.17.13 Tawk cannot find the column names... What is the problem?

First, make sure the comment char (`-s` option) is correctly set for your file (the default is `'%'`). Second, make sure the column names do not contain forbidden characters, i.e., use only alphanumeric and underscore and do not start with a number. If the row with column names is not the last one to start with the separator character, then specify the line number with the `-N` option as follows: `tawk -N 3'` or `tawk -s '#' -N 2`. Refer to Section 3.5 for more details.

3.17.14 Wireshark refuses to open PCAP files generated with tawk `-k` option...

If Wireshark displays the message `Couldn't run /usr/bin/dumpcap in child process: Permission Denied.`, then this means that your user does not belong to the `wireshark` group. To fix this issue, simply run the following command `sudo gpasswd -a YOUR_USERNAME wireshark` (you will then need to log off and on again).

3.17.15 Tawk reports errors similar to `free(): double free detected in tcache 2`

Tawk uses `gawk -M` option to handle IPv6 addresses. For some reasons, this option is regularly affected by bugs... If you do not need IPv6 support, you can simply comment out line 653 in `tawk`:

```
OPTS=(
    #-M -v PREC=256         # <-- Add the leading sharp ('#') here
    -v __PRIHDR__=$PRIHDR
    -v __UNAME__="$(uname)"
)
```

3.17.16 Tawk `-k` reports errors similar to `Couldn't run dumpcap in child process: Permission denied`

On some Linux distributions, capturing packets is only allowed as `root` or as a member of the `wireshark` group.

Run the following command to add yourself to the `wireshark` group.

```
$ sudo usermod -a -G wireshark $USER
```

Then, log out and log in again (or reboot) and the problem should be fixed!

More information on the Wireshark wiki.