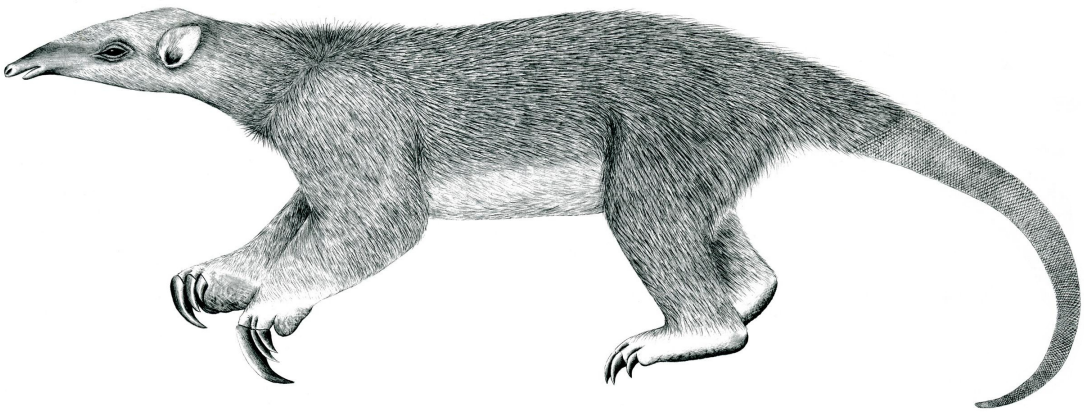
# Tranalyzer2

## tawk

Awk for Tranalyzer Flow Files



Tranalyzer Development Team

# Contents

# 1 tawk

## 1.1 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.

## 1.2 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**[1]**:** | `sudo dnf install gawk` |
| **macOS**[2]**:** | `brew install gawk` |

## 1.3 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/tranalyzer2-0.9.0`):

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

### 1.3.1 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.

## 1.4 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`

[1] If the `dnf` command could not be found, try with `yum` instead
[2] 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 42: `tawk -x flow42.pcap '$flowInd == 42' file_flows.txt`
- To create a PCAP with packets 4-10: `tawk -P -x pkts-4_to_10.pcap 'packet("4-10")' file_flows.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_flows.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`.

## 1.5 -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 ''`.

## 1.6 Related Utilities

### 1.6.1 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`

### 1.6.2 lsx

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

### 1.6.3 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`

### 1.6.4 tcol

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

## 1.7 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`.<br>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`.<br>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`.<br>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`<br>(comma or semicolon separated).<br>Ranges may also be specified using a dash, e.g., `port("1-3")`.<br>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).<br>Ranges may also be specified using a dash, e.g., `dport("1-3")`.<br>If `p` is omitted, return the destination port. |
| `sport([p])` | Return true if the source port appears in `p` (comma or semicolon separated).<br>Ranges may also be specified using a dash, e.g., `sport("1-3")`.<br>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).<br>Ranges may also be specified using a dash, e.g., `proto("1-3")`.<br>If `p` is omitted, return the protocol number. |
| `proto2str([p])` | Return the string representation of the protocol number `p`.<br>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.<br>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.<br>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`.<br>If `num` is omitted or 0, `val` is split into `osep` separated columns.<br>If `num > 0`, return the `num` repetition.<br>If `num < 0`, return the `num` repetition from the end, e.g., -1 for last element.<br>Multiple `num` can be specified, e.g., `"1;-1;2"`.<br>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.<br>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. |
| 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. |
| unquote(s) | Remove leading and trailing quotes from a string. |
| chomp(s) | Remove leading and trailing spaces from a string. |
| strip(s) | Remove leading and trailing spaces from a string. |
| lstrip(s) | Remove leading spaces from a string. |
| rstrip(s) | Remove trailing spaces from a string. |
| mean(c) | Compute the mean value of a column c.<br>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.<br>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.<br>The result can be accessed with get_max(c) or printed with print_max([c]). |
| abs(v) | Return the absolute value of v. |
| min2(a,b) | Return the minimum value between a and b. |
| min3(a,b,c) | Return the minimum value between a, b and c. |
| max2(a,b) | Return the maximum value between a and b. |
| max3(a,b,c) | Return the maximum value between a, b and c. |

| Function | Description |
|---|---|
| `aggr(fields[,val[,num]])` | Perform aggregation of `fields` and store the sum of `val`.<br>`fields` and `val` can be tab separated lists of fields, e.g., `$srcIP4"\t"$dstIP4`.<br>Results are sorted according to the first value of `val`.<br>If `val` is omitted, the empty string or equal to `"flows"` or `"packets"`<br>(case insensitive), count the number of records (flows or packets).<br>If `num` is omitted or 0, return the full list.<br>If `num > 0` return the top `num` results.<br>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`.<br>`val` can be a tab separated lists of fields, e.g., `$numBytesSnt"\t"$numPktsSnt`.<br>Results are sorted according to the first value of `val`.<br>If `val` is omitted, the empty string or equal to `"flows"` or `"packets"`<br>(case insensitive), count the number of records (flows or packets).<br>If `num` is omitted or 0, return the full list.<br>If `num > 0` return the top `num` results.<br>If `num < 0` return the bottom `num` results.<br>If `ign_e` is omitted or 0, consider all values, otherwise ignore empty values.<br>`sep` can be used to change the separator character (default: `";"`). |
| `t2rsort(col[,num[,type]])` | Sort the file in reverse order according to `col`.<br>(Multiple column numbers can be specified by using `";"` as separator,<br>e.g., `1 ";" 2`)<br>If `num` is omitted or 0, return the full list.<br>If `num > 0` return the top `num` results.<br>If `num < 0` return the bottom `num` results.<br>`type` can be used to specify the type of data to sort:<br>`"ip"`, `"num"` or `"str"` (default is based on the first matching record). |
| `t2sort(col[,num[,type[,rev]]])` | Sort the file according to `col`.<br>(Multiple column numbers can be specified by using `";"` as separator,<br>e.g., `1 ";" 2`)<br>If `num` is omitted or 0, return the full list.<br>If `num > 0` return the top `num` results.<br>If `num < 0` return the bottom `num` results.<br>`type` can be used to specify the type of data to sort:<br>`"ip"`, `"num"` or `"str"` (default is based on the first matching record).<br>If `rev > 0`, sort in reverse order (alternatively, use the `t2rsort()` function). |
| `t2whois(ip[,o_opt])` | Wrapper to call `t2whois` from `tawk`.<br>`ip` must be a valid IPv4/6 address.<br>`o_opt` is passed verbatim to `t2whois -o` option<br>(run `t2whois -L` for more details). |
| `wildcard(expr)` | Print all columns whose name matches the regular expression `expr`. |

| Function | Description |
|---|---|
| | 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 its 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`. |
| `printerr(s)` | Print the string `s` in red with an added newline. |
| `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 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`). 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`. |

| Function | Description |
|---|---|
| | of can be used to change the output format [default: 0]:<br>0: Payload only,<br>1: prefix each payload with packet/flow info,<br>2: JSON,<br>3: Reconstruct (pipe the output to xxd -p -r to reproduce the binary file).<br>d can be used to only extract a specific direction ("A" or "B")<br>[default: "" (A and B)].<br>pf can be used to change the payload format [default: 0]:<br>0: ASCII,<br>1: Hexdump,<br>2: Raw/Binary,<br>3: Base64.<br>r can be used to prevent the analysis of TCP sequence numbers<br>(no TCP reassembly and reordering).<br>nc can be used to print the data without colors. |
| shark(q) | Query flow files according to Wireshark's syntax. |

## 1.8 Examples

Collection of examples using tawk functions:

| Function | Description |
|---|---|
| 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).<br>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.<br>The following information is reported for each reply:<br>FirstSeen, LastSeen, Type (A or AAAA), TTL, Query, Answer,<br>Organization, Country, AS number |
| passwords([val[,num]]) | Return information about hosts sending authentication in cleartext.<br>If val is omitted or equal to "flows", count the number of flows.<br>Otherwise, sum up the values of val.<br>If num is omitted or 0, returns the full list.<br>If num > 0 return the top num results.<br>If num < 0 return the bottom num results. |
| postQryStr([n]) | Return the top N POST requests with query strings. |

| Function | Description |
|---|---|
| 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, ...). |

## 1.9 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 |

| Function | Description |
|---|---|
| `mpls10()` | MPLS label 10 |
| `mpls()` | MPLS labels 1–10 |
| `bps()` | Bits per second |
| `pps()` | Packets per second |
| `bpp()` | Bytes per package |
| | |
| `oline()` | nfdump line output format (`-o line`) |
| `olong()` | nfdump long output format (`-o long`) |
| `oextended()` | nfdump extended output format (`-o extended`) |

## 1.10 t2custom

Copy your own functions in this folder. Refer to Section 1.11 for more details on how to write a tawk function. To have your functions automatically loaded, include them in the file `t2custom/t2custom.load`.

## 1.11 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
    }
}
```

## 1.12 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`

## 1.13 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 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 no column names are present, use the `-t` option to prevent tawk from trying to validate the column names.
- If the column names are different from those used by Tranalyzer, refer to Section 1.13.1.

### 1.13.1 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.

### 1.13.2 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
```

## 1.14 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.

### 1.14.1 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`.

### 1.14.2 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
```

## 1.15 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 $srcIP4, $dstIP4 }' 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 $srcIP4, $dstIP4 }' file.txt`
  - `awkf '`

```
NR == 1 {
    for (i = 1; i <= NF; i++) {
        if ($i == "srcIP4") srcIP4 = i
        else if ($i == "dstIP4") dstIP4 = i
    }
    if (srcIP4 == 0 || dstIP4 == 0) {
        print "No column with name srcIP4 and/or dstIP4"
        exit
```

```
            }
        }
        NR > 1 {
            print $srcIP4, $dstIP4
        }
    ' file.txt

- awkf '
        NR == 1 {
            for (i = 1; i <= NF; i++) {
                col[$i] = i
            }
        }
        NR > 1 {
            print $col["srcIP4"], $col["dstIP4"];
        }

    ' file.txt
```

## 1.16 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($srcIP4 "\t" $dstIP4, $numBytesSnt, 10)' FILE_flows.txt

   tawk 'aggr(tuple2(), $numBytesSnt "\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($srcIP4, "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($srcIP4, "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($srcIP4, "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($srcIP4, "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($srcIP4, "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`.

## 1.17 FAQ

### 1.17.1 Can I use tawk with non Tranalyzer files?

Yes, refer to Section 1.13.

### 1.17.2 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 1.13.1 for more details.

### 1.17.3 Can I use tawk with files without column names?

Yes, use the `-t` option to prevent tawk from trying to validate the column names.

### 1.17.4 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'`

### 1.17.5 Can I process Bro/Zeek log files with tawk?

Yes, use the `--zeek` option.

### 1.17.6 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 1.5).

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

### 1.17.7 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`

### 1.17.8 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`

### 1.17.9 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'`

### 1.17.10 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 1.5 for more details.

### 1.17.11 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.

### 1.17.12 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 1.5 for more details.

### 1.17.13 How to make tawk faster?

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. This verification process is quite slow and can easily by disabled by using the `-t` option.

### 1.17.14 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).

### 1.17.15 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)"
)
```