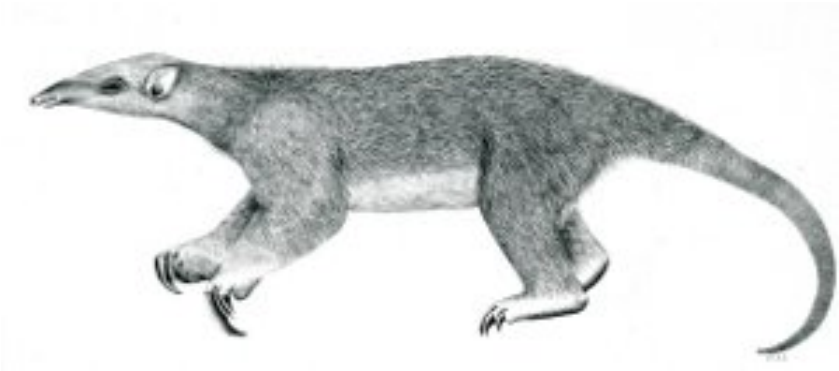
Tranalyzer2

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.

Kali/Ubuntu: `sudo apt-get install gawk`

Arch: `sudo pacman -S gawk`

Fedora/Red Hat: `sudo yum install gawk`

Gentoo: `sudo emerge gawk`

OpenSUSE: `sudo zypper install gawk`

Mac OS X: `brew install gawk`[1]

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/int_tranalyzer/trunk`):

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

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`
- 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($dnsStat)' file_flows.txt`
- To consult the documentation for the function 'func': `tawk -d func`
- To consult the documentation for the functions 'min' and 'max': `tawk -d min,max`
- To consult the documentation for all the available functions: `tawk -d all`
- To consult the documentation for the variable 'var': `tawk -V var`
- To consult the documentation for the variable 'var' with value `0x8a`: `tawk -V var=0x8a`
- To convert the output to JSON: `tawk '{ print json($flowStat "\t" tuple5()) }' file_flows.txt`
- To convert the output to JSON: `tawk 'aggr(tuple2())' file_flows.txt | tawk '{ print json($0) }'`
- To create a PCAP with all packets from flow 42: `tawk -x flow42.pcap '$flowInd == 42' file_flows.txt`
- To see all ICMP packets in Wireshark: `tawk -k 'imcp()' 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. Some of the most useful are:

- Changing the input field separator:
  `tawk -F ',' '!hdr() { print $colName }' file.csv`
- Changing the output field separator:
  `tawk -v OFS=',' '!hdr() { print $col1, $col2 }' file.txt`

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

1.5 `-s` Option

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 behaviour, the line number can be specified as well. For example if row 1 to 5 start with `'#'` and row 3 contains the column names, specify the separator as follows: `tawk -s '#NR==3'` If the row with column names does not start with a special character, use `-s ''` or `-s 'NR==2'`.

1.6 Related Utilities

1.6.1 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`

1.6.2 lsx

Displays 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 occuring rows. (Alias for `sort | uniq -c | sort -rn`). Useful, e.g., to analyse the most occuring user-agents: `tawk '{ print $httpUsrAg }' FILE_flows.txt | sortu`

1.6.4 tcol

Displays 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()`	Returns the 2 tuple (source IP and destination IP)
`tuple3()`	Returns the 3 tuple (source IP, destination IP and port)
`tuple4()`	Returns the 4 tuple (source IP and port, destination IP and port)
`tuple5()`	Returns the 5 tuple (source IP and port, destination IP and port, protocol)
`tuple6()`	Returns the 6 tuple (source IP and port, dest. IP and port, proto, VLANID)
`host([ip|net])`	Returns true if the source or destination IP is equal to `ip` or belongs to `net` If `ip` is omitted, returns the source and destination IP
`shost([ip|net])`	Returns true if the source IP is equal to `ip` or belongs to `net`

Function	Description
	If ip is omitted, returns the source IP
dhost([ip|net])	Returns true if the destination IP is equal to ip or belongs to net If ip is omitted, returns 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)	Returns true if ip is a loopback address
mcast(ip)	Returns true if ip is a multicast address
privip(ip)	Returns true if ip is a private IP
port([p])	Returns true if the source or destination port is equal to p (multiple ports or port ranges can also be specified) If p is omitted, returns the source and destination port
sport([p])	Returns true if the source port is equal to p If p is omitted, returns the source port
dport([p])	Returns true if the destination port is equal to p If p is omitted, returns the destination port
ipv4()	Returns true if the flow contains IPv4 traffic
ipv6()	Returns true if the flow contains IPv6 traffic
proto([p])	Returns true if the protocol is equal to p If p is omitted, returns the string representation of the protocol
proto2str(p)	Returns the string representation of the protocol number p If p is omitted, returns the protocol
icmp([p])	Returns true if the protocol is equal to 1 (ICMP)
igmp([p])	Returns true if the protocol is equal to 2 (IGMP)
tcp([p])	Returns true if the protocol is equal to 6 (TCP)
udp([p])	Returns true if the protocol is equal to 17 (UDP)
rsvp([p])	Returns true if the protocol is equal to 46 (RSVP)
gre([p])	Returns true if the protocol is equal to 47 (GRE)
esp([p])	Returns true if the protocol is equal to 50 (ESP)
ah([p])	Returns true if the protocol is equal to 51 (AH)
icmp6([p])	Returns true if the protocol is equal to 58 (ICMPv6)
sctp([p])	Returns true if the protocol is equal to 132 (SCTP)
dhcp()	Returns true if the flow contains DHCP traffic
dns()	Returns true if the flow contains DNS traffic
http()	Returns true if the flow contains HTTP traffic
tcpflags([val])	If val is specified, returns true if the specified flags are set. If val is omitted, returns a string representation of the TCP flags
ip2num(ip)	Converts an IP address to a number

Function	Description
`ip2hex(ip)`	Converts an IPv4 address to hex
`ip2str(ip)`	Converts an IPv4 address to string
`ip62str(ip)`	Converts an IPv6 address to string
`ip6compress(ip)`	Compresses an IPv6 address
`ip6expand(ip[,trim])`	Expands an IPv6 address. If `trim` is different from 0, removes leading zeros
`ip2mask(ip)`	Converts an IP address to a network mask (int)
`mask2ip(m)`	Converts a network mask (int) to an IPv4 address (int)
`mask2ipstr(m)`	Converts a network mask (int) to an IPv4 address (string)
`mask2ip6(m)`	Converts a network mask (int) to an IPv6 address (int)
`mask2ip6str(m)`	Converts a network mask (int) to an IPv6 address (string)
`ipinnet(ip,net[,mask])`	Tests whether an IP address belongs to a given network
`ipinrange(ip,low,high)`	Tests whether an IP address lies between two addresses
`localtime(t)`	Converts UNIX timestamp to string (localtime)
`utc(t)`	Converts UNIX timestamp to string (UTC)
`timestamp(t)`	Converts date to UNIX timestamp
`t2split(val,sep [,num[,osep]])`	Splits values according to `sep`. If `num` is omitted or 0, `val` is split into `osep` separated columns. If `num > 0`, returns the `num` repetition. If `num < 0`, returns 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]])`	Splits compound values. Alias for `t2split(val, "_", num, osep)`
`splitr(val[,num[,osep]])`	Splits repetitive values. Alias for `t2split(val, ";", num, osep)`
`valcontains(val,sep,item)`	Returns 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)`	Returns true if `val` is an empty string
`streq(val1,val2)`	Returns true if `val1` is equal to `val2`
`strneq(val1,val2)`	Returns true if `val1` and `val2` are not equal
`hasprefix(val,pre)`	Returns true if `val` begins with the prefix `pre`
`hassuffix(val,suf)`	Returns true if `val` finished with the suffix `suf`
`contains(val,txt)`	Returns true if `val` contains the substring `txt`
`not(q)`	Returns the logical negation of a query `q`. This function must be used to keep the header when negating a query.
`bfeq(val1,val2)`	Returns true if the bitfields (hexadecimal numbers) `val1` and `val2` are equal
`bitsallset(val,mask)`	Returns true if all the bits set in `mask` are also set in `val`
`bitsanyset(val,mask)`	Returns true if one of the bits set in `mask` is also set in `val`

Function	Description
bitisset(val,mask[,mode])	*If mode is omitted or 0, returns true if all the bits set in* mask *are also set in* val *else returns true if one of the bits set in* mask *is also set in* val **This function has been deprecated (replaced by** bitsallset **and** bitsanyset**)**
isip(v)	Returns true if v is an IPv4 address in hexadecimal, numerical or dotted decimal notation
isip6(v)	Returns true if v is an IPv6 address
isiphex(v)	Returns true if v is an IPv4 address in hexadecimal notation
isipnum(v)	Returns true if v is an IPv4 address in numerical (int) notation
isipstr(v)	Returns true if v is an IPv4 address in dotted decimal notation
isnum(v)	Returns true if v is a number
join(a,s)	Converts an array to string, separating each value with s
unquote(s)	Removes leading and trailing quotes from a string
chomp(s)	Removes leading and trailing spaces from a string
strip(s)	Removes leading and trailing spaces from a string
lstrip(s)	Removes leading spaces from a string
rstrip(s)	Removes trailing spaces from a string
abs(v)	Returns the absolute value of v
mean(c)	Computes the mean value of a column c. The result can be accessed with get_mean(c) or printed with print_mean([c])
min(a,b)	Returns the minimum value between a and b
min3(a,b,c)	Returns the minimum value between a, b and c
max(a,b)	Returns the maximum value between a and b
max3(a,b,c)	Returns the maximum value between a, b and c
aggr(fields[,val[,num]])	Performs aggregation of fields and store the sum of val. fields and val can be tab separated lists of fields, e.g., $srcIP4"\t"$dstIP4 Results are sorted according to the first value of val. If val is omitted or equal to "flows", counts the number of flows. If num is omitted or 0, returns the full list, If num > 0 returns the top num results, If num < 0 returns the bottom num results.
aggrrep(fields[,val[,num[,ign_e[,sep]]]])	Performs aggregation of the repetitive fields and store the sum of val. val can be a tab separated lists of fields, e.g., $numBytesSnt"\t"$numPktsSnt Results are sorted according to the first value of val. If val is omitted or equal to "flows", counts the number of flows. If num is omitted or 0, returns the full list, If num > 0 returns the top num results, If num < 0 returns the bottom num results. If ign_e is omitted or 0, consider all values, otherwise ignore emtpy values. sep can be used to change the separator character (default: ";")
t2sort(col[,num[,type]])	Sorts the file according to col.

Function	Description
	If num is omitted or 0, returns the full list, If num > 0 returns the top num results, If num < 0 returns 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)
wildcard(expr)	Print all columns whose name matches the regular expression expr. If expr is preceded by an exclamation mark, returns 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.
texscape(s)	Escape the string s to make it LaTeX compatible
base64d(s)	Decode a base64 encoded string s
urldecode(url)	Decode the encoded URL url
printerr(s)	Prints the string s in red with an added newline
diff(file[,mode])	Compares file and the input, and prints the name 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, creates 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)	Returns all flows whose index appears in f
packet(p)	Returns all packets whose number appears in f
shark(q)	Query flow files according to Wireshark's syntax

1.8 Examples

Collection of examples using tawk functions:

Function	Description
covertChans([val[,num]])	Returns information about hosts possibly involved in a covert channels. If val is omitted or equal to "flows", counts the number of flows. Otherwise, sums up the values of val. If num is omitted or 0, returns the full list, If num > 0 returns the top num results, If num < 0 returns the bottom num results.
dnsZT()	Returns all flows where a DNS zone transfer was performed.

Function	Description
exeDL([n])	Returns the top N EXE downloads.
httpHostsURL([f])	Returns all HTTP hosts and a list of the files hosted (sorted alphabetically). If f > 0, prints the number of times a URL was requested.
nonstdports()	Returns all flows running protocols over non-standard ports.
passwords([val[,num]])	Returns information about hosts sending authentication in cleartext. If val is omitted or equal to "flows", counts the number of flows. Otherwise, sums up the values of val. If num is omitted or 0, returns the full list, If num > 0 returns the top num results, If num < 0 returns the bottom num results.
postQryStr([n])	Returns the top N POST requests with query strings.
ssh()	Returns the SSH connections.
topDnsA([n])	Returns the top N DNS answers.
topDnsIp4([n])	Returns the top N DNS answers IPv4 addresses.
topDnsIp6([n])	Returns the top N DNS answers IPv6 addresses.
topDnsQ([n])	Returns the top N DNS queries.
topHttpMimesST([n])	Returns the top HTTP content-type (type/subtype).
topHttpMimesT([n])	Returns the top HTTP content-type (type only).
topSLD([n])	Returns the top N second-level domains queried (google.com, yahoo.com, ...).
topTLD([n])	Returns 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

Function	Description
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 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 Files

To use tawk with Bro log files, use the following command:

```
tawk -s '#fields' -i $T2HOME/scripts/tawk/vars_bro 'hdr() || !/^#/ { 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
- `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 'bitisset($3,1)' 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 a CSV (Comma Separated Value) file with tawk?

The input field separator can be changed with the `-F` option. To process a CSV file, run tawk as follows:
`tawk -F ',' 'program' file.csv`

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

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

1.17.7 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`

1.17.8 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'`

1.17.9 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.

1.17.10 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.

1.17.11 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 (`NR`) as follows: `-s '#NR==3'` or `-s '%NR==2'`. Refer to Section 1.5 for more details.