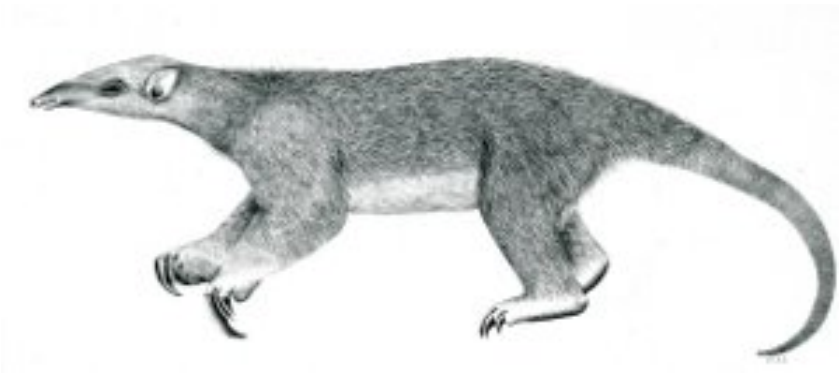
Tranalyzer2

Version 0.8.4, Beta, Tarantula

Flow based forensic and network troubleshooting traffic analyzer



Tranalyzer Development Team

Contents

1 Introduction

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

1.1 Getting Tranalyzer

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

1.2 Dependencies

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

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

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

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

Gentoo: `sudo emerge autoconf automake libpcap libtool zlib`

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

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

1.3 Compilation

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

- Tranalyzer2 only:
  `cd "$T2HOME"; ./autogen.sh tranalyzer2`
  (alternative: `cd "$T2HOME/tranalyzer2"; ./autogen.sh`)
- A specific plugin only, e.g., `myPlugin`:
  `cd "$T2HOME"; ./autogen.sh myPlugin`
  (alternative 1: `cd "$T2PLHOME/myPlugin"; ./autogen.sh`)
  (alternative 2: `cd "$T2HOME/plugins/myPlugin"; ./autogen.sh`)
- Tranalyzer2 and a default set of plugins:
  `cd "$T2HOME"; ./autogen.sh`

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

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

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

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

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

1.3.1 Custom Build

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

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

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

# Build the txtSink plugin
txtSink
```

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

1.4 Installation

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

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

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

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

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

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

1.4.1 Aliases

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

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

1.5 Getting Started

Run Tranalyzer as follows:

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

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

1.6 Getting Help

1.6.1 Documentation

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

1.6.2 Man Page

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

```
man tranalyzer
```

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

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

1.6.3 Help

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

1.6.4 FAQ

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

1.6.5 Contact

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

tranalyzer@rdit.ch

2 Tranalyzer2

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

2.1 Supported Link-Layer Header Types

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

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

2.2 Enabling/Disabling Plugins

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

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

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

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

2.2.1 Default

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

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

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

2.2.2 Whitelisting Plugins

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

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

```
# This is a comment

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

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

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

2.2.3 Blacklisting Plugins

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

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

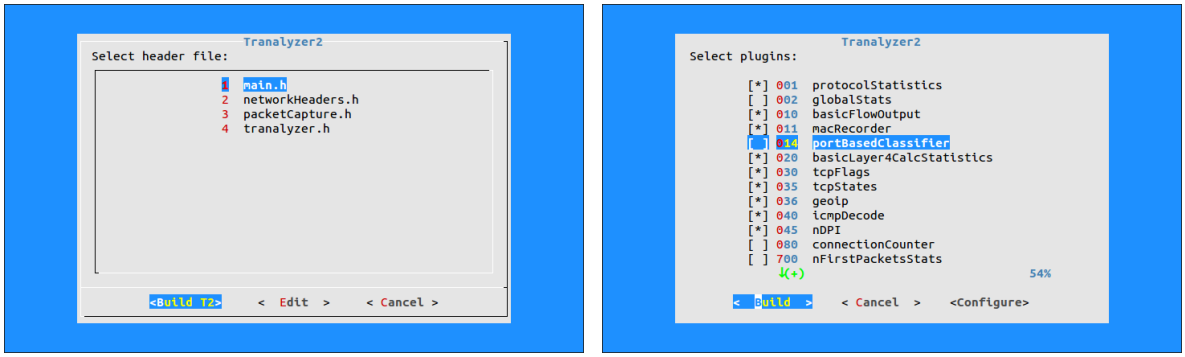
```
# This is a comment

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

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

2.2.4 Graphical Configuration and Building of T2 and Plugins

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



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

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

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

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

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

2.3 Man Page

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

```
man tranalyzer
```

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

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

2.4 Invoking Tranalyzer

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

2.4.1 Help

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

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

Usage:
```

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

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

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

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

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

    -v           Show the version of the program and exit

    -h           Show help options and exit

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

2.4.2 –i INTERFACE

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

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

2.4.3 –r FILE

Capture data from a pcap file `FILE`.

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

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

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

2.4.4 –R FILE

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

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

2.4.5 –D FILE[*][.ext]#1[:SCHR][,#2]

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

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

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

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

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

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

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

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

Or to process files 10 to 100:

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

Or to process files 10 to 100 in another format:

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

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

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

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

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

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

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

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

2.4.6 –w PREFIX

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

2.4.7 –W PREFIX[:SIZE][,START]

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

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

2.4.8 –l

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

2.4.9 –s

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

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

```
    q=0.3\r\nAccept-Encoding: gzip,deflate\r\nAccept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\
    nKeep-Alive: 115\r\nConnection: keep-alive\r\nReferer: http://z-ecx.images-amazon.com/images/I
    /11J5cf408UL.css\r\n\r\n
...
```

2.4.10 –p FOLDER

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

2.4.11 –b FILE

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

2.4.12 –e FLOWINDEXFILE

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

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

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

2.4.13 –f HASHFACTOR

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

2.4.14 –x SENSORID

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

2.4.15 –c CPU

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

2.4.16 –F FILE

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

2.4.17 BPF Filter

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

2.5 ioBuffer.h

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

2.6 main.h

The monitoring mode can be configured with the following constants:

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

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

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

2.7 networkHeaders.h

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

2.8 packetCapture.h

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

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

2.9 tranalyzer.h

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

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

2.9.1 -D constants

the following constants influence the file name convention:

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

2.9.2 alive signal

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

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

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

2.9.3 FORCE_MODE

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

2.9.4 ALARM_MODE

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

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

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

2.9.5 BLOCK_BUF

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

2.9.6 Report Modes

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

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

The following interrupts are being caught by Tranalyzer2:

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

2.9.7 State and statistical save mode

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

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

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

2.9.8 L2TP

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

2.9.9 GRE

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

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

2.9.10 FRAGMENTATION

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

2.9.11 FRAG_HLST_CRFT

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

2.9.12 FRAG_ERROR_DUMP

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

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

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

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

2.9.13 *_SUFFIX

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

2.9.14 RELTIME

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

2.9.15 FLOW_TIMEOUT

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

2.9.16 FDURLIMIT

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

2.9.17 HASHFACTOR

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

2.9.18 HASHTABLE_SIZE

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

2.9.19 HASHCHAINTABLE_SIZE

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

2.9.20 HASH_AUTOPILOT

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

2.9.21 Aggregation Mode

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

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

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

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

2.10 bin2txt.h

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

2.11 outputBuffer.h

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

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

2.12 Tranalyzer2 Output

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

2.12.1 Hierarchical Ordering of Numerical or Text Output

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

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

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

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

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

2.13 Final Report

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

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

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

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

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

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

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

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

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

2.14 Monitoring Modes During Runtime

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

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

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

(MACHINE_REPORT=0) is shown in Listing 2.

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

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

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

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

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

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

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

2.14.1 Configuration for Monitoring Mode

To enable monitoring mode, configure Tranalyzer as follows:

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

The following plugins contribute to the output:

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

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

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

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

- PktsRec
- PktsDrp
- IfDrp

2.14.2 Monitoring Mode to syslog

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

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

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

2.14.3 RRD Graphing of Monitoring Output

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

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

Or for monitoring from a live interface:

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

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

An example graph is depicted in Figure 3.

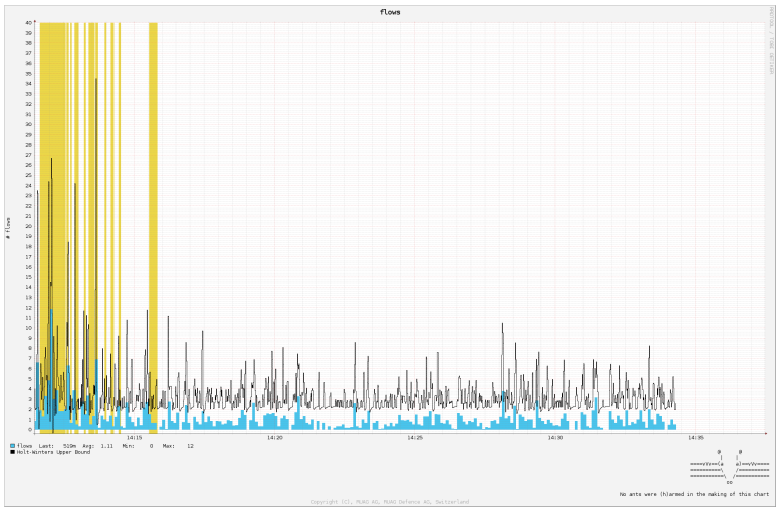


Figure 3: *T2 monitoring using RRD*

2.15 Cancellation of the Sniffing Process

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

3 arpDecode

3.1 Description

The arpDecode plugin analyzes ARP traffic.

3.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
MAX_IP	10	Max. number of MAC/IP pairs to list

3.3 Flow File Output

The arpDecode plugin outputs the following columns:

Column	Type	Description
arpStat	H8	Status
arpHwType	U16	Hardware type
arpOpcode	H16	Operational code
arpIpMacCnt	U16	Number of distinct MAC / IP pairs
arpMac_Ip_Cnt	MAC_IP4_U16	MAC, IP pairs found and number of times the pair appeared. (a count of zero may appear in case of ARP spoofing and indicates the pair was discovered in a different flow)

3.3.1 arpStat

The arpStat column is to be interpreted as follows:

arpStat	Description
0x01	ARP detected
0x02	MAC/IP list truncated... increase MAX_IP
0x08	Gratuitous ARP (sender IP same as target IP)
0x80	ARP spoofing (same IP assigned to multiple MAC)

3.3.2 arpHwType

The `arpHwType` column is to be interpreted as follows:

Type	Description
1	Ethernet
2	Experimental Ethernet
3	Amateur Radio AX.25
4	Proteon ProNET Token Ring
5	Chaos
6	IEEE 802
7	ARCNET
8	Hyperchannel
9	Lanstar
10	Autonet Short Address
11	LocalTalk
12	LocalNet (IBM PCNet or SYTEK LocalNET)
13	Ultra link
14	SMDS
15	Frame Relay
16	ATM (Asynchronous Transmission Mode)
17	HDLC
18	Fibre Channel

Type	Description
19	ATM (Asynchronous Transmission Mode)
20	Serial Line
21	ATM (Asynchronous Transmission Mode)
22	MIL-STD-188-220
23	Metricom
24	IEEE 1394.1995
25	MAPOS
26	Twinaxial
27	EUI-64
28	HIPARP
29	IP and ARP over ISO 7816-3
30	ARPSec
31	IPsec tunnel
32	Infiniband
33	CAI (TIA-102 Project 25 Common Air Interface)
34	Wiegand Interface
35	Pure IP

3.3.3 arpOpcode

The `arpOpcode` column is to be interpreted as follows:

arpOpcode	Description
2^0 (=0x0001)	—
2^1 (=0x0002)	ARP Request
2^2 (=0x0004)	ARP Reply
2^3 (=0x0008)	Reverse ARP (RARP) Request
2^4 (=0x0010)	Reverse ARP (RARP) Reply
2^5 (=0x0020)	Dynamic RARP (DRARP) Request
2^6 (=0x0040)	Dynamic RARP (DRARP) Reply
2^7 (=0x0080)	Dynamic RARP (DRARP) Error

arpOpcode	Description
2^8 (=0x0100)	Inverse ARP (InARP) Request
2^9 (=0x0200)	Inverse ARP (InARP) Reply
2^{10} (=0x0400)	ARP NAK
2^{11} (=0x0800)	—
2^{12} (=0x1000)	—
2^{13} (=0x2000)	—
2^{14} (=0x4000)	—
2^{15} (=0x8000)	—

3.4 Plugin Report Output

The following information is reported:

- Aggregated status flags (arpStat)

3.5 Packet File Output

In packet mode (-s option), the arpDecode plugin outputs the following columns:

Column	Description
arpStat	Status
arpHwType	Hardware type
arpProtoType	Protocol type
arpHwSize	Hardware size
arpProtoSize	Protocol size
arpOpcode	Operational code
arpSenderMAC	Sender MAC address
arpSenderIP	Sender IP address
arpTargetMAC	Target MAC address
arpTargetIP	Target IP address

4 basicFlow

4.1 Description

The basicFlow plugin provides host identification fields and timing information.

4.2 Configuration Flags

4.2.1 basicFlow.h

The following flags can be used to control the output of the plugin:

Name	Default	Description	Flags
BFO_SENSORID	0	1: sensorID on / 0: sensorID off	
BFO_HDRDESC_PKTCNT	0	1: Enables / 0: Disables pkt count for header description	
BFO_MAC	1	1: Enables / 0: Disables MAC address output	
BFO_ETHERTYPE	1	1: Enables / 0: Disables EtherType output	IPV6_ACTIVATE=2|| ETH_ACTIVATE>0
BFO_VLAN	1	0: Do not output VLAN information, 1: Output VLAN numbers, 2: Output VLAN headers as hex	
BFO_MPLS	0	0: Do not output MPLS information, 1: Output MPLS labels, 2: Output MPLS headers as hex, 3: Output decoded MPLS headers	
BFO_L2TP	0	1: Enables L2TP header information	
BFO_GRE	0	1: Enables GRE header information	
BFO_PPP	0	1: Enables PPP header information	
BFO_ETHIP	0	1: Enables ETHIP header information	
BFO_TEREDO	0	1: Enables Teredo IP, Port information	
BFO_SUBNET_TEST	1	1: Enables subnet test	
BFO_SUBNET_TEST_GRE	0	1: Enable subnet test on GRE addresses	IPV6_ACTIVATE!=1
BFO_SUBNET_TEST_L2TP	0	1: Enables subnet test on L2TP addresses	IPV6_ACTIVATE!=1
BFO_SUBNET_TEST_TEREDO	0	1: Enables subnet test on Teredo addresses	
BFO_SUBNET_HEX	0	Country code and who information representation: 0: Two human readable columns (two letters country code and who), 1: One column, hex ID output	
BFO_SUBNET_ASN	0	1: Autonomous System Numbers on, 0: ASN off	
BFO_SUBNET_LL	0	1: Latitude, longitude and reliability, 0: no output	
BFO_MAX_HDRDESC	4	Max. number of headers descriptions to store 0: switch off output	T2_PRI_HDRDESC=1
BFO_MAX_MAC	2	Max. number of different MAC addresses to store	

		0: switch off output
BFO_MAX_MPLS	3	Max. number of MPLS Header pointer to store 0: switch off output
BFO_MAX_VLAN	3	Max. number of Ethertypes to store 0: switch off output

4.2.2 utils.h

The following flags can be used to control the output of the plugin: If SUBRNG or WHOEN is changed, the basicFlow plugin

Name	Default	Description
SUBRNG	0	Subnet definition 1: Begin - End / 0: CIDR only
WHOLEN	20	length of WHO record

MUST be recompiled with `'./autogen.sh -f'`.

4.2.3 bin2txt.h

Additional configuration options can be found in $T2HOME/utils/bin2txt.h. Refer to Tranalyzer2 documentation for more details.

4.3 Flow File Output

The basicFlow plugin outputs the following columns:

Column	Type	Description	Flags
dir	C	Flow direction A / B	
flowInd	U64	Flow index	
sensorID	U32	Sensor ID	BFO_SENSORID=1
flowStat	H64	Flow status and warnings	
timeFirst	TS	Date time of first packet	
timeLast	TS	Date time of last packet	
duration	U64.U32	Flow duration	

If T2_PRI_HDRDESC=1 and BFO_HDRDESC_DEPTH>0, the following columns are displayed:

numHdrDesc	U8	Number of different headers descriptions	
numHdrs	RU16	Number of headers (depth) in hdrDesc	BFO_HDRDESC_PKTCNT=1
hdrDesc_PktCnt	RS_U64	Headers description and packet count	
srcMac	R(MAC)	Source MAC address	BFO_MAC=1
dstMac	R(MAC)	Destination MAC address	BFO_MAC=1
ethType	H16	Ethernet type	BFO_ETHERTYPE=1&& (ETH_ACTIVATE>0|| IPV6_ACTIVATE=2)

Column	Type	Description	Flags
If BFO_VLAN>0 and BFO_MAX_VLAN_DEPTH>0, the following column is displayed:			
ethVlanID	U16R	VLAN IDs	BFO_VLAN=1
ethVlanHdr	RH32	VLAN headers (hex)	BFO_VLAN=2
If BFO_MPLS>0 and BFO_MAX_MPLS_DEPTH>0, the following column is displayed:			
mplsLabels	RU32	MPLS labels	BFO_MPLS=1
mplsTagsHex	RH32	MPLS tags (hex)	BFO_MPLS=2
mplsLabel_ToS_S_TTL	R(U32_U8_U8_U8)	MPLS tags detail	BFO_MPLS=3
If BFO_PPP=1, the following column is displayed:			
pppHdr	H32	PPP header	
If BFO_L2TP=1, the following columns are displayed:			
l2tpHdr	H16	L2TP header	
l2tpTID	U16	L2TP tunnel ID	
l2tpSID	U16	L2TP session ID	
l2tpCCSID	U32	L2TP control connection/session ID	
l2tpSrcIP	IP4	L2TP source IP address	
l2tpSrcIPASN	U32	L2TP source IP ASN	BFO_SUBNET_TEST_L2TP=1&&BFO_SUBNET_ASN=1
l2tpSrcIPCC	S/H32	L2TP source IP country code	BFO_SUBNET_TEST_L2TP=1
l2tpSrcIPWho	S	L2TP source IP organisation name	BFO_SUBNET_TEST_L2TP=1&&BFO_SUBNET_HEX=0
l2tpSrcIPLat_Lng_relP	F_F_F	L2TP source IP latitude, longitude and reliability	BFO_SUBNET_TEST_L2TP=1&&BFO_SUBNET_LL=1
l2tpDstIP	IP4	L2TP destination IP address	
l2tpDstIPASN	U32	L2TP destination IP ASN	BFO_SUBNET_TEST_L2TP=1&&BFO_SUBNET_ASN=1
l2tpDstIPCC	S/H32	L2TP destination IP country code	BFO_SUBNET_TEST_L2TP=1
l2tpDstIPWho	S	L2TP destination IP organisation name	BFO_SUBNET_TEST_L2TP=1&&BFO_SUBNET_HEX=0
l2tpDstIPLat_Lng_relP	F_F_F	L2TP destination IP latitude, longitude and reliability	BFO_SUBNET_TEST_L2TP=1&&BFO_SUBNET_LL=1
If BFO_GRE=1, the following columns are displayed:			
greHdr	H32	GRE header	
greSrcIP	IP4	GRE source IP address	
greSrcIPASN	U32	GRE source IP ASN	BFO_SUBNET_TEST_GRE=1&&BFO_SUBNET_ASN=1

Column	Type	Description	Flags
greSrcIPCC	S/H32	GRE source IP country code	BFO_SUBNET_TEST_GRE=1
greSrcIPWho	S	GRE source IP organisation name	BFO_SUBNET_TEST_GRE=1&& BFO_SUBNET_HEX=0
greSrcIPLat_Lng_relP	F_F_F	GRE source IP latitude, longitude and reliability	BFO_SUBNET_TEST_GRE=1&& BFO_SUBNET_LL=1
greDstIP	IP4	GRE destination IP address	
greDstIPASN	U32	GRE destination IP ASN	BFO_SUBNET_TEST_GRE=1&& BFO_SUBNET_ASN=1
greDstIPCC	S/H32	GRE destination IP country code	BFO_SUBNET_TEST_GRE=1
greDstIPWho	S	GRE destination IP organisation name	BFO_SUBNET_TEST_GRE=1&& BFO_SUBNET_HEX=0
greDstIPLat_Lng_relP	F_F_F	GRE destination IP latitude, longitude and reliability	BFO_SUBNET_TEST_GRE=1&& BFO_SUBNET_LL=1
If BFO_TEREDO=1, the following columns are displayed:			
trdoDstIP	IP4	Nxt Teredo Flow: Dest IPv4 address	
trdoDstIPASN	U32	Teredo destination IP ASN	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_ASN=1
trdoDstIPCC	S/H32	Teredo destination IP country code	BFO_SUBNET_TEST_TEREDO=1
trdoDstIPWho	S	Teredo destination IP organisation name	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_HEX=0
trdoDstIPLat_Lng_relP	F_F_F	Teredo destination IP latitude, longitude and reliability	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_LL=1
trdoDstPort	U16	Nxt Teredo Flow: Destination port	
If BFO_TEREDO=1 and IPV6_ACTIVATE>0 then the following lines are displayed:			
trdo6SrcFlgs	H8	Teredo IPv6 source address decode: Flags	
trdo6SrcSrvIP4	IP4	Teredo IPv6 source address decode: Server IPv4	
trdo6SrcSrvIP4ASN	U32	Teredo IPv6 source address decode: Server IPv4 ASN	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_ASN=1
trdo6SrcSrvIP4CC	S/H32	Teredo IPv6 source address decode: Server IPv4 country code	BFO_SUBNET_TEST_TEREDO=1
trdo6SrcSrvIP4Who	S	Teredo IPv6 source address decode: Server IPv4 who	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_HEX=0
trdo6SrcSrvIP4Lat_Lng_relP	F_F_F	Teredo IPv6 source address decode: Server IPv4 latitude, longitude and reliability	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_LL=1
trdo6SrcCPIP4	IP4	Teredo IPv6 source address decode: Client Public IPv4	
trdo6SrcCPIP4ASN	U32	Teredo IPv6 source address decode: Client Public IPv4 ASN	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_ASN=1
trdo6SrcCPIP4CC	S/H32	Teredo IPv6 source address decode:	BFO_SUBNET_TEST_TEREDO=1

Column	Type	Description	Flags
		Client Public IPv4 country code	
trdo6SrcCPIP4Who	S	Teredo IPv6 source address decode: Client Public IPv4 who	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_HEX=0
trdo6SrcCPIP4Lat_Lng_relP	F_F_F	Teredo IPv6 source address decode: Client Public IPv4 latitude, longitude and reliability	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_LL=1
trdo6SrcCPPort	U16	Teredo IPv6 source address decode: Client Public Port	
trdo6DstFlgs	H8	Teredo IPv6 dest. address decode: Flags	
trdo6DstSrvIP4	IP4	Teredo IPv6 dest. address decode: Server IPv4	
trdo6DstSrvIP4ASN	U32	Teredo IPv6 dest. address decode: Server IPv4 ASN	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_ASN=1
trdo6DstSrvIP4CC	S/H32	Teredo IPv6 dest. address decode: Server IPv4 country code	BFO_SUBNET_TEST_TEREDO=1
trdo6DstSrvIP4Who	S	Teredo IPv6 dest. address decode: Server IPv4 who	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_HEX=0
trdo6DstSrvIP4Lat_Lng_relP	F_F_F	Teredo IPv6 dest. address decode: Server IPv4 latitude, longitude and reliability	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_LL=1
trdo6DstCPIP4	IP4	Teredo IPv6 dest. address decode: Client Public IPv4	
trdo6DstCPIP4ASN	U32	Teredo IPv6 dest. address decode: Client Public IPv4 ASN	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_ASN=1
trdo6DstCPIP4CC	S/H32	Teredo IPv6 dest. address decode: Client Public IPv4 country code	BFO_SUBNET_TEST_TEREDO=1
trdo6DstCPIP4Who	S	Teredo IPv6 dest. address decode: Client Public IPv4 who	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_HEX=0
trdo6DstCPIP4Lat_Lng_relP	F_F_F	Teredo IPv6 dest. address decode: Client Public IPv4 latitude, longitude and reliability	BFO_SUBNET_TEST_TEREDO=1&& BFO_SUBNET_LL=1
trdo6DstCPPort	U16	Teredo IPv6 dest. address decode: Client Public Port	

Standard six tuple output including geolabeling:

srcIP	IP	Source IP address	
srcIPASN	U32	Source IP ASN	BFO_SUBNET_TEST=1&& BFO_SUBNET_ASN=1
srcIPCC	S/H32	Source IP country code	BFO_SUBNET_TEST=1
srcIPWho	S	Source IP organisation name	BFO_SUBNET_TEST=1&& BFO_SUBNET_HEX=0
srcIPLat_Lng_relP	F_F_F	Source IP latitude, longitude and reliability	BFO_SUBNET_TEST=1&& BFO_SUBNET_LL=1
srcPort	U16	Source Port	

Column	Type	Description	Flags
dstIP4	IP	Destination IP address	
dstIPASN	U32	Destination IP ASN	BFO_SUBNET_TEST=1&& BFO_SUBNET_ASN=1
dstIPCC	S/H32	Destination IP country code	BFO_SUBNET_TEST=1
dstIPWho	S	Destination IP organisation name	BFO_SUBNET_TEST=1 BFO_SUBNET_HEX=0
dstIPLat_Lng_relP	F_F_F	Destination IP latitude, longtable and reliability	BFO_SUBNET_TEST=1&& BFO_SUBNET_LL=1
dstPort	U16	Destination port	
l4Proto	U8	Layer 4 protocol	

4.3.1 flowInd

It is useful to identify flows when post processing operations, such as sort or filters are applied to a flow file and only a B or an A flow is selected. Moreover a packet file generated with the -s option supplies the flow index which simplifies the mapping of singular packets to the appropriate flow.

4.3.2 flowStat

The flowStat column is to be interpreted as follows:

flowStat	Description
2^{00} (=0x00000000 00000001)	Inverted Flow, did not initiate connection
2^{01} (=0x00000000 00000002)	No Ethernet header
2^{02} (=0x00000000 00000004)	Pure L2 Flow
2^{03} (=0x00000000 00000008)	Point to Point Protocol over Ethernet Discovery (PPPoED)
2^{04} (=0x00000000 00000010)	Point to Point Protocol over Ethernet Service (PPPoES)
2^{05} (=0x00000000 00000020)	Link Layer Discovery Protocol (LLDP)
2^{06} (=0x00000000 00000040)	ARP
2^{07} (=0x00000000 00000080)	Reverse ARP
2^{08} (=0x00000000 00000100)	VLANs
2^{09} (=0x00000000 00000200)	MPLS unicast
2^{10} (=0x00000000 00000400)	MPLS multicast
2^{11} (=0x00000000 00000800)	L2TP v2/3
2^{12} (=0x00000000 00001000)	GRE v1/2
2^{13} (=0x00000000 00002000)	PPP header after L2TP or GRE
2^{14} (=0x00000000 00004000)	IPv4
2^{15} (=0x00000000 00008000)	IPv6
2^{16} (=0x00000000 00010000)	IPvX bogus packets

	flowStat	Description
2^{17}	(=0x00000000 00020000)	IPv4/6 in IPv4/6
2^{18}	(=0x00000000 00040000)	Ethernet over IP
2^{19}	(=0x00000000 00080000)	Teredo tunnel
2^{20}	(=0x00000000 00100000)	Anything in Anything (AYIYA) Tunnel
2^{21}	(=0x00000000 00200000)	GPRS Tunneling Protocol (GTP)
2^{22}	(=0x00000000 00400000)	Virtual eXtensible Local Area Network (VXLAN)
2^{23}	(=0x00000000 00800000)	Control and Provisioning of Wireless Access Points (CAPWAP), Lightweight Access Point Protocol (LWAPP)
2^{24}	(=0x00000000 01000000)	Stream Control Transmission Protocol (SCTP)
2^{25}	(=0x00000000 02000000)	SSDP/UPnP
2^{26}	(=0x00000000 04000000)	Encapsulated Remote Switch Packet ANalysis (ERSPAN)
2^{27}	(=0x00000000 08000000)	Cisco Web Cache Communication Protocol (WCCP)
2^{28}	(=0x00000000 10000000)	SIP/RTP
2^{29}	(=0x00000000 20000000)	Generic Network Virtualization Encapsulation (GENEVE)
2^{30}	(=0x00000000 40000000)	Authentication Header (AH)
2^{31}	(=0x00000000 80000000)	—
2^{32}	(=0x00000001 00000000)	Acquired packet length < minimal L2 datagram
2^{33}	(=0x00000002 00000000)	Acquired packet length < packet length in L3 header
2^{34}	(=0x00000004 00000000)	Acquired packet length < minimal L3 Header
2^{35}	(=0x00000008 00000000)	Acquired packet length < minimal L4 Header
2^{36}	(=0x00000010 00000000)	IPv4 fragmentation present
2^{37}	(=0x00000020 00000000)	IPv4 fragmentation error (refer to the tcpFlags plugin for more details)
2^{38}	(=0x00000040 00000000)	IPv4 1. fragment out of sequence or missing
2^{39}	(=0x00000080 00000000)	Fragmentation sequence not completed when flow timeout
2^{40}	(=0x00000100 00000000)	Flow timeout instead of protocol termination
2^{41}	(=0x00000200 00000000)	Alarm mode: remove this flow instantly
2^{42}	(=0x00000400 00000000)	Autopilot: Flow removed to free space in main hash map
2^{43}	(=0x00000800 00000000)	Stop dissecting, error or not capable to do e.g. IPv4/6 config
2^{44}	(=0x00001000 00000000)	PPPL3 header not readable, compressed
2^{45}	(=0x00002000 00000000)	—
2^{46}	(=0x00004000 00000000)	—
2^{47}	(=0x00008000 00000000)	—
2^{48}	(=0x00010000 00000000)	Header description overrun
2^{49}	(=0x00020000 00000000)	pcapd and PD_ALARM=1: if set dumps the packets from this flow to a new pcap
2^{50}	(=0x00040000 00000000)	Land attack: same srcIP && dstIP && srcPort && dstPort
2^{51}	(=0x00080000 00000000)	Time slip possibly due to NTP operations on the capture machine
2^{52}	(=0x00100000 00000000)	liveXtr : if set dumps the packets from this flow to a new pcap

flowStat	Description
2^{56} (=0x01000000 00000000)	Tor address detected
2^{57} (=0x02000000 00000000)	A packet had a priority tag (VLAN tag with ID 0)
2^{63} (=0x80000000 00000000)	PCAP packet length > MAX_MTU in *ioBuffer.h*, caplen reduced

4.3.3 hdrDesc

The hdrDesc column describes the protocol stack in the flow in a human readable way. Note that it gives the user a lookahead of what is to be expected, even if not in the appropriate IPv4/6 mode. For example, in IPv4 several different headers stacks can be displayed by one flow if Teredo or different fragmentation is involved. T2 then dissects only to the last header above the said protocol and sets the *Stop dissecting* bit in the flow status (2^{41} (=0x00000400 00000000)).

4.3.4 trdoFlags

The trdoFlags column is to be interpreted as follows:

trdoFlags	Description
2^0 (=0x01)	Group/individual
2^1 (=0x02)	Universal/local
2^2 (=0x04)	0
2^3 (=0x08)	0
2^4 (=0x10)	0
2^5 (=0x20)	0
2^6 (=0x40)	Currently Unassigned
2^7 (=0x80)	Behind Nat, new version do not set this bit anymore

4.3.5 Geo labeling

The country coding scheme is defined in utils/cntrycd.txt. The special values [0-9][0-9] are used to represent private addresses or special address ranges such as teredo or multicast:

- 00: 10.0.0.0/8 (private)
- 01: 172.16.0.0/16 (private)
- 02: 192.168.0.0/16 (private)
- 03: 169.254.0.0/16 (link-local)
- 04: 224.0.0.0/8 (multicast)
- 01: fe80::/10 (link local)
- 02: fc00::/7 (private)
- 03: ::ffff:0.0.0.0/96
- 04: ff00::/8 (multicast)
- 10: 2001::/32 (teredo)

The text format of the subnets4.txt and subnets6.txt files is defined as follows:
A '-' in the first column (prefix/mask) denotes a non-CIDR range. In this case, Tranalyzer reads the 2nd column instead of the 1st when SUBRNG=1 in utils.h. If SUBRNG=0, the 2nd column is ignored and only CIDR ranges are accepted.

The text files subnets4.txt and subnets6.txt can be edited and manually converted as follows:

```
#       3       20190114
# prefix/mask   seMask  start_ip-end_ip coCode  asn     probability     long    lat
    country_code    organisation
10.0.0.0/8      8       10.0.0.0-10.255.255.255 0x01003690      0       1.000000        666.000000
          666.000000      00      private_reserved
172.16.0.0/12   12      172.16.0.0-172.31.255.255       0x01003690      0       1.000000
    666.000000      666.000000      01      private_reserved
192.168.0.0/16  16      192.168.0.0-192.168.255.255     0x01003690      0       1.000000
    666.000000      666.000000      02      private_reserved
169.254.0.0/16  16      169.254.0.0-169.254.255.255     0x01003690      0       1.000000
    666.000000      666.000000      03      private_reserved
224.0.0.0/8     8       224.0.0.0-224.255.255.255       0x01002c68      0       1.000000
    666.000000      666.000000      04      Multicast
1.0.0.0/24      24      1.0.0.0-1.0.0.255       0x0e000000      0       0.980000        145.179990
          -37.700000      au      regional internet registry for the asia-pacific region
1.0.1.0/24      24      1.0.1.0-1.0.1.255       0x31000000      0       0.970000        666.000000
          666.000000      cn      chinanet fujian province network
1.0.1.0/24      22      1.0.1.0-1.0.3.255       0x31000000      0       0.980000        119.309990
          26.059990       cn      chinanet fujian province network
1.0.100.0/22    22      1.0.100.0-1.0.103.255   0x73000000      0       0.980000        133.050000
          35.470000       jp      --
-   22      9.111.0.15-9.112.2.116   0x54000000      0       0.980000        13.050000
    225.470000      us      IBM
......
```

`./utils/subconv -4 subnets4.txt` and `./utils/subconv -6 subnets6.txt`

4.4 Packet File Output

In packet mode (`-s` option), the basicFlow plugin outputs the following columns:

Column	Description	Flags
flowInd	Flow index	
flowStat	Flow status	
time	Time	
relTime	Duration since start of pcap or interface sniffing	
pktIAT	Packet inter-arrival time	
flowDuration	Flow duration	
numHdrs	Number of headers (depth) in hdrDesc	T2_PRI_HDRDESC=1
hdrDesc	Headers description	T2_PRI_HDRDESC=1
ethVlanID	VLAN number (inner VLAN)	
srcMac	Source MAC address	
dstMac	Destination MAC address	
ethType	Ethernet type	
srcIP	Source IP address	
srcIPCC	Source IP country code	BFO_SUBNET_TEST=1
srcIPWho	Source IP organisation name	BFO_SUBNET_TEST=1
srcPort	Source port	
dstIP	Destination IP address	
dstIPCC	Destination IP country code	BFO_SUBNET_TEST=1

Column	Description	Flags
dstIPWho	Destination IP organisation name	BFO_SUBNET_TEST=1
dstPort	Destination port	
l4Proto	Layer 4 protocol	

5 basicStats

5.1 Description

The basicStats plugin supplies basic layer four statistics for each flow.

5.2 Dependencies

5.2.1 Other Plugins

If the basicFlow plugin is loaded, then the country of the IPs with the most bytes and packets transmitted is displayed in the final report.

5.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description	Flags
BS_AGGR_CNT	0	1: add A+B counts, 0: A+B counts off	
BS_REV_CNT	1	1: add reverse counts from opposite flow, 0: native send counts	
BS_STATS	1	Output statistics (min, max, average, ...)	
BS_PL_STATS	1	1: Packet Length statistics	
BS_IAT_STATS	1	1: IAT statistics	
If BS_STATS==1, the following additional flags can be used:			
BS_VAR	0	Output the variance	
BS_STDDEV	1	Output the standard deviation	
BS_XCLD	0	0: do not exclude any value from statistics, 1: include (BS_XMIN,UINT16_MAX], 2: include [0,BS_XMAX), 3: include [BS_XMIN,BS_XMAX] 4: exclude (BS_XMIN,BS_XMAX)	
BS_XMIN	1	minimal included/excluded from statistics	BS_XCLD>0
BS_XMAX	65535	maximal included/excluded from statistics	BS_XCLD>0

5.4 Flow File Output

The basicStats plugin outputs the following fields:

Column	Type	Description	Flags
numPktsSnt	U64	Number of transmitted packets	
numPktsRcvd	U64	Number of received packets	BS_REV_CNT=1
numPktsRTAggr	U64	Number of received + transmitted packets	BS_AGGR_CNT=1
numBytesSnt	U64	Number of transmitted bytes	

Column	Type	Description	Flags
numBytesRcvd	U64	Number of received bytes	BS_REV_CNT=1
numBytesRTAggr	U64	Number of received + transmitted bytes	BS_AGGR_CNT=1
If BS_STATS=1, the following columns, whose value depends on BS_XCLD, are provided			
If BS_PL_STATS=1, the following five columns are displayed			
minPktSz	U16	Minimum layer 3 packet size	
maxPktSz	U16	Maximum layer 3 packet size	
avePktSize	F	Average layer 3 packet size	
varPktSize	F	Variance layer 3 packet size	BS_VAR=1
stdPktSize	F	Standard deviation layer 3 packet size	BS_STDDEV=1
If BS_IAT_STATS=1, the following five columns are displayed			
minIAT	F	Minimum IAT	
maxIAT	F	Maximum IAT	
aveIAT	F	Average IAT	
varIAT	F	Variance IAT	BS_VAR=1
stdIAT	F	Standard deviation IAT	BS_STDDEV=1
pktps	F	Sent packets per second	
bytps	F	Sent bytes per second	
pktAsm	F	Packet stream asymmetry	
bytAsm	F	Byte stream asymmetry	

5.5 Packet File Output

In packet mode (-s option), the basicFlow plugin outputs the following columns:

Column	Description
pktLen	Packet size on the wire
l7Len	L7 length

5.6 Plugin Report Output

The IP of biggest packets/bytes talker and packets/bytes counts are reported.

6 binSink

6.1 Description

The binSink plugin is one of the basic output plugin for Tranalyzer2. It uses the output prefix (`-w` option) to generate a binary flow file with suffix `_flows.bin`. All standard output from every plugin is stored in binary format in this file.

6.2 Dependencies

6.2.1 External Libraries

If gzip compression is activated (`GZ_COMPRESS=1`), then **zlib** must be installed.

Kali/Ubuntu: `sudo apt-get install zlib1g-dev`

Arch: `sudo pacman -S zlib`

Fedora/Red Hat: `sudo yum install zlib-devel`

Gentoo: `sudo emerge zlib`

OpenSUSE: `sudo zypper install zlib-devel`

Mac OS X: `brew install zlib`[4]

6.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
`GZ_COMPRESS`	0	Compress the output (gzip)
`SFS_SPLIT`	1	Split the output file (Tranalyzer `-W` option)
`FLOWS_SUFFIX`	`"_flows.bin"`	Suffix to use for the output file
`STD_BUFSHFT`	`BUF_DATA_SHFT * 4`	

6.4 Post-Processing

6.4.1 tranalyzer-b2t

The program `tranalyzer-b2t` can be used to transform binary Tranalyzer files into text or json files. The converted file uses the same format as the one generated by the txtSink or jsonSink plugin.

The program can be found in `$T2HOME/utils/tranalyzer-b2t/` and can be compiled by typing `make`.

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

The use of the program is straightforward:

- bin→txt: `./tranalyzer-b2t -r FILE_flows.bin -w FILE_flows.txt`
- bin→json: `./tranalyzer-b2t -r FILE_flows.bin -j -w FILE_flows.json`

If the `-w` option is omitted, the destination default to stdout.
Additionally, the `-n` option can be used **not** to print the name of the columns as the first row.

6.5 Custom File Output

- `PREFIX_flows.bin`: Binary representation of Tranalyzer output

7 cdpDecode

7.1 Description

The cdpDecode plugin analyzes CDP traffic.

7.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
CDP_STRLEN	512	Maximum length of strings to store

7.3 Flow File Output

The cdpDecode plugin outputs the following columns:

Column	Type	Description
cdpStat	H8	Status
cdpVersion	U8	Version
cdpTTL	U8	Time To Live (sec)
cdpTLVTypes	H32	Aggregated TLV types
cdpDevice	SC	Device ID
cdpPlatform	S	Platform
cdpPort	SC	Port ID
cdpCaps	H32	Capabilities
cdpDuplex	H8	Duplex
cdpNVLAN	U16	Native VLAN
cdpVTPMngmtDomain	SC	VTP management domain

7.3.1 cdpStat

The cdpStat column is to be interpreted as follows:

cdpStat	Description
0x01	Flow is CDP
0x80	Snapped payload

7.3.2 cdpTLVTypes

The `cdpTLVTypes` column is to be interpreted as follows:

cdpTLVTypes	Description
2^0 (=0x0000 0001)	—
2^1 (=0x0000 0002)	Device ID
2^2 (=0x0000 0004)	Addresses
2^3 (=0x0000 0008)	Port ID
2^4 (=0x0000 0010)	Capabilities
2^5 (=0x0000 0020)	Software Version
2^6 (=0x0000 0040)	Platform
2^7 (=0x0000 0080)	IP Prefixes
2^8 (=0x0000 0100)	Protocol Hello
2^9 (=0x0000 0200)	VTP Management Domain
2^{10} (=0x0000 0400)	Native VLAN
2^{11} (=0x0000 0800)	Duplex
2^{12} (=0x0000 1000)	—
2^{13} (=0x0000 2000)	—
2^{14} (=0x0000 4000)	—
2^{15} (=0x0000 8000)	VoIP VLAN Query

cdpTLVTypes	Description
2^{16} (=0x0001 0000)	Power Consumption
2^{13} (=0x0002 0000)	—
2^{18} (=0x0004 0000)	Trust Bitmap
2^{19} (=0x0008 0000)	Untrusted Port CoS
2^{20} (=0x0010 0000)	—
2^{21} (=0x0020 0000)	—
2^{22} (=0x0040 0000)	Management Address
2^{23} (=0x0080 0000)	—
2^{24} (=0x0100 0000)	—
2^{25} (=0x0200 0000)	Power Requested
2^{26} (=0x0400 0000)	Power Available
2^{27} (=0x0800 0000)	—
2^{28} (=0x1000 0000)	—
2^{29} (=0x2000 0000)	—
2^{30} (=0x4000 0000)	—
2^{31} (=0x8000 0000)	Any type ≥ 31

7.3.3 cdpCaps

The `cdpCaps` column is to be interpreted as follows:

cdpCaps	Description
0x0000 0001	Router
0x0000 0002	Transparent Bridge
0x0000 0004	Source Route Bridge
0x0000 0008	Switch
0x0000 0010	Host
0x0000 0020	IGMP capable
0x0000 0040	Repeater
0x00000100-0x80000000	Reserved

7.3.4 cdpDuplex

The `cdpDuplex` column is to be interpreted as follows:

cdpDuplex	Description
0x0001	Half
0x0002	Full

7.4 Plugin Report Output

The following information is reported:

- Number of CDP packets

8 connStat

8.1 Description

The connStat plugin counts the connections between different IPs and ports per flow and during the pcap lifetime in order to produce an operational picture for anomaly detection.

8.2 Dependencies

8.2.1 Other Plugins

If the basicFlow plugin is loaded, then the country of the IPs with the most connections is displayed in the final report.

8.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
CS_HSDRM	1	decrement IP counters when flows die
CS_SDIPMAX	1	0: number of src dst IP connnections 1: IP src dst connection with the highest count

8.4 Flow File Output

The connStat plugin outputs the following columns:

Column	Type	Description
connSip	U32	Number of unique source IPs
connDip	U32	Number of unique destination IPs
connSipDip	U32	Number of connections between source and destination IPs
connSipDprt	U32	Number of connections between source IP and destination port
connF	F	the f number, experimental: connSipDprt/connSip

8.5 Plugin Report Output

The following information is reported:

- Number of unique source IPs
- Number of unique destination IPs
- Number of unique source/destination IPs connections
- Max unique number of source IP / destination port connections
- IP prtcon/sdcon, prtcon/scon
- Source IP with the max connections
- Destination IP with max connections

9 descriptiveStats

9.1 Description

The descriptiveStats plugin calculates various statistics about a flow. Because the inter-arrival time of the first packet is per definition always zero, it is removed from the statistics. Therefore the inter-arrival time statistics values for flows with only one packet is set to zero.

9.2 Dependencies

9.2.1 Other Plugins

This plugin requires the pktSIATHisto plugin.

9.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
ENABLE_PS_CALC	1	1: Enables / 0: Disables calculation of statistics for packet sizes
ENABLE_IAT_CALC	1	1: Enables / 0: Disables calculation of statistics for inter-arrival times

9.4 Flow File Output

The descriptiveStats plugin outputs the following columns:

Column	Type	Description	Flags
MinPl	F	Minimum packet length	ENABLE_PS_CALC=1
MaxPl	F	Maximum packet length	ENABLE_PS_CALC=1
MeanPl	F	Mean packet length	ENABLE_PS_CALC=1
LowQuartilePl	F	Lower quartile of packet lengths	ENABLE_PS_CALC=1
MedianPl	F	Median of packet lengths	ENABLE_PS_CALC=1
UppQuartilePl	F	Upper quartile of packet lengths	ENABLE_PS_CALC=1
IqdPl	F	Inter quartile distance of packet lengths	ENABLE_PS_CALC=1
ModePl	F	Mode of packet lengths	ENABLE_PS_CALC=1
RangePl	F	Range of packet lengths	ENABLE_PS_CALC=1
StdPl	F	Standard deviation of packet lengths	ENABLE_PS_CALC=1
RobStdPl	F	Robust standard deviation of packet lengths	ENABLE_PS_CALC=1
SkewPl	F	Skewness of packet lengths	ENABLE_PS_CALC=1
ExcPl	F	Excess of packet lengths	ENABLE_PS_CALC=1
MinIat	F	Minimum inter-arrival time	ENABLE_IAT_CALC=1
MaxIat	F	Maximum inter-arrival time	ENABLE_IAT_CALC=1
MeanIat	F	Mean inter-arrival time	ENABLE_IAT_CALC=1
LowQuartileIat	F	Lower quartile of inter-arrival times	ENABLE_IAT_CALC=1
MedianIat	F	Median of inter-arrival times	ENABLE_IAT_CALC=1
UppQuartileIat	F	Upper quartile of inter-arrival times	ENABLE_IAT_CALC=1

Column	Type	Description	Flags
IqdIat	F	Inter quartile distance of inter-arrival times	ENABLE_IAT_CALC=1
ModeIat	F	Mode of inter-arrival times	ENABLE_IAT_CALC=1
RangeIat	F	Range of inter-arrival times	ENABLE_IAT_CALC=1
StdIat	F	Standard deviation of inter-arrival times	ENABLE_IAT_CALC=1
RobStdIat	F	Robust standard deviation of inter-arrival times	ENABLE_IAT_CALC=1
SkewIat	F	Skewness of inter-arrival times	ENABLE_IAT_CALC=1
ExcIat	F	Excess of inter-arrival times	ENABLE_IAT_CALC=1

9.5 Known Bugs and Limitations

Because the packet length and inter-arrival time plugin stores the inter-arrival times in statistical bins the original time information is lost. Therefore the calculation of the inter-arrival times statistics is due to its logarithmic binning only a rough approximation of the original timing information. Nevertheless, this representation has shown to be useful in practical cases of anomaly and application classification.

10 dhcpDecode

10.1 Description

This dhcpDecode plugin analyzes DHCP traffic.

10.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description	Flags
DHCPBITFLD	1	Options representation: 1: bitfield, 0: option numbers in a row	
DHCPMAXOPT	50	maximum stored options	DHCPBITFLD=0
DHCPNMMAX	10	maximal number of domain/host names per flow	
DHCPMASKFRMT	1	Netmask representation: 0: hex, 1: IP	
DHCP_ADD_CNT	0	Print the number of times a given mac/domain/host appeared	
DHCP_FLAG_MAC	0	Store a global mapping IP->MAC and add the source and destination MAC address to every flow [EXPERIMENTAL]	
DHCP_FM_DEBUG	0	print debug information about DHCP_FLAG_MAC operations	

10.3 Flow File Output

The dhcpDecode plugin outputs the following columns:

Column	Type	Description	Flags
dhcpStat	H16	Status, warnings and errors	
dhcpMType	H16/H32	Message type	
dhcpHWType	H32	Hardware Type	
dhcpCHWAdd	R(MAC)	Client hardware addresses	DHCP_ADD_CNT=0
dhcpCHWAdd_HWCnt	R(MAC_H32)	Client hardware addresses and count	DHCP_ADD_CNT=1
If IPV6_ACTIVATE == 0|2, the following columns are output:			
dhcpNetmask	H32/IP4	Network mask	DHCPMASKFRMT=0/1
dhcpGWIP	IP4	Gateway IP	
dhcpDnsIP	IP4	DNS IP	
dhcpHopCnt	H32	Hop Count	
dhcpSrvName	S	Server host name	
dhcpBootFile	S	Boot file name	
dhcpOptCnt	U16	Option Count	
dhcpOpts	RU8	Options	DHCPBITFLD=0
dhcpOptBF1_BF2_BF3	H64_H64_H64	Option Bit field	DHCPBITFLD=1
dhcpHosts	R(S)	Maximal DHCPNMMAX hosts	DHCP_ADD_CNT=0
dhcpHosts_HCnt	R(S_U16)	Maximal DHCPNMMAX hosts and count	DHCP_ADD_CNT=1
dhcpDomains	R(S)	Maximal DHCPNMMAX domains	DHCP_ADD_CNT=0
dhcpDomains_DCnt	R(S_U16)	Maximal DHCPNMMAX domains and count	DHCP_ADD_CNT=1
dhcpMaxSecEl	U16	Maximum seconds elapsed	

Column	Type	Description	Flags
dhcpLeaseT	U32	Lease time	
dhcpRenewT	U32	Renewal time	
dhcpRebindT	U32	Rebind time	
dhcpCliIP	IP4	DHCP client IP	
dhcpYourIP	IP4	DHCP your (client) IP	
dhcpNextServer	IP4	DHCP next server IP	
dhcpRelay	IP4	DHCP relay agent IP	
dhcpLFlow	U64	DHCP linked flow	
dhcpSrcMac	MAC	DHCP source MAC address	DHCP_FLAG_MAC=1
dhcpDstMac	MAC	DHCP destination MAC address	DHCP_FLAG_MAC=1

10.3.1 dhcpStat

The dhcpStat status bit field is to be interpreted as follows:

dhcpStat	Description
0x0001	DHCP detected
0x0002	Boot request
0x0004	Boot reply
0x0008	Broadcast
0x0010	Client ID (option 61) different from Client MAC address
0x0020	Option overload: server host name and/or boot file name carry options
0x0100	Option list truncated... increase DHCPMAXOPT
0x0200	Client HW address, domain or host name list truncated... increase DHCPNMMAX
0x2000	Error: DHCP magic number corrupt
0x4000	Error: DHCP options corrupt
0x8000	Something weird happened...

10.3.2 dhcpMType

For IPv4, the dhcpMType column is to be interpreted as follows:

dhcpMType4	Description
2^1 (=0x0002)	Discover Message
2^2 (=0x0004)	Offer Message
2^3 (=0x0008)	Request Message
2^4 (=0x0010)	Decline Message
2^5 (=0x0020)	Acknowledgment Message
2^6 (=0x0040)	Negative Acknowledgment Message
2^7 (=0x0080)	Release Message
2^8 (=0x0100)	Informational Message

For IPv6, the dhcpMType column is to be interpreted as follows:

dhcpMType6	Description
0x0000 0001	Reserved
0x0000 0002	SOLICIT
0x0000 0004	ADVERTISE
0x0000 0008	REQUEST
0x0000 0010	CONFIRM
0x0000 0020	RENEW
0x0000 0040	REBIND
0x0000 0080	REPLY
0x0000 0100	RELEASE
0x0000 0200	DECLINE
0x0000 0400	RECONFIGURE
0x0000 0800	INFORMATION-REQUEST

dhcpMType6	Description
0x0000 1000	RELAY-FORW
0x0000 2000	RELAY-REPL
0x0000 4000	LEASEQUERY
0x0000 8000	LEASEQUERY-REPLY
0x0001 0000	LEASEQUERY-DONE
0x0002 0000	LEASEQUERY-DATA
0x0004 0000	RECONFIGURE-REQUEST
0x0008 0000	RECONFIGURE-REPLY
0x0010 0000	DHCPV4-QUERY
0x0020 0000	DHCPV4-RESPONSE
0x0040 0000	ACTIVELEASEQUERY
0x0080 0000	STARTTLS

10.3.3 dhcpHWType

The dhcpHWType column is to be interpreted as follows:

dhcpHWType	Description
2^0 (=0x0000 0000 0000 0001)	—
2^1 (=0x0000 0000 0000 0002)	Ethernet
2^2 (=0x0000 0000 0000 0004)	Experimental Ethernet
2^3 (=0x0000 0000 0000 0008)	Amateur Radio AX.25
2^4 (=0x0000 0000 0000 0010)	Proteon ProNET Token Ring
2^5 (=0x0000 0000 0000 0020)	Chaos
2^6 (=0x0000 0000 0000 0040)	IEEE 802
2^7 (=0x0000 0000 0000 0080)	ARCNET
2^8 (=0x0000 0000 0000 0100)	Hyperchannel
2^9 (=0x0000 0000 0000 0200)	Lanstar
2^{10} (=0x0000 0000 0000 0400)	Autonet Short Address
2^{11} (=0x0000 0000 0000 0800)	LocalTalk
2^{12} (=0x0000 0000 0000 1000)	LocalNet (IBM PCNet or SYTEK LocalNET)
2^{13} (=0x0000 0000 0000 2000)	Ultra link
2^{14} (=0x0000 0000 0000 4000)	SMDS
2^{15} (=0x0000 0000 0000 8000)	Frame Relay
2^{16} (=0x0000 0000 0001 0000)	ATM, Asynchronous Transmission Mode
2^{17} (=0x0000 0000 0002 0000)	HDLC
2^{18} (=0x0000 0000 0004 0000)	Fibre Channel
2^{19} (=0x0000 0000 0008 0000)	ATM, Asynchronous Transmission Mode
2^{20} (=0x0000 0000 0010 0000)	Serial Line
2^{21} (=0x0000 0000 0020 0000)	ATM, Asynchronous Transmission Mode
2^{22} (=0x0000 0000 0040 0000)	MIL-STD-188-220
2^{23} (=0x0000 0000 0080 0000)	Metricom
2^{24} (=0x0000 0000 0100 0000)	IEEE 1394.1995
2^{25} (=0x0000 0000 0200 0000)	MAPOS

dhcpHWType	Description
2^{26} (=0x0000 0000 0400 0000)	Twinaxia
2^{27} (=0x0000 0000 0800 0000)	EUI-64
2^{28} (=0x0000 0000 1000 0000)	HIPARP
2^{29} (=0x0000 0000 2000 0000)	IP and ARP over ISO 7816-3
2^{30} (=0x0000 0000 4000 0000)	ARPSec
2^{31} (=0x0000 0000 8000 0000)	IPsec tunnel
2^{32} (=0x0000 0001 0000 0000)	Infiniband
2^{33} (=0x0000 0002 0000 0000)	CAI, TIA-102 Project 25 Common Air Interface
2^{34} (=0x0000 0004 0000 0000)	Wiegand Interface
2^{35} (=0x0000 0008 0000 0000)	Pure IP
2^{63} (=0x8000 0000 0000 0000)	All values bigger than 62 are reported here

10.3.4 dhcpHopCnt

The dhcpHopCnt column is to be interpreted as follows:

dhcpHopCnt	Description
0x00000000-0x00010000	Number of hops (0–16) $\left(2^{\text{HopCount}}\right)$
0x80000000	Invalid hop count (> 16)

10.3.5 dhcpOptBF1_BF2_BF3

The dhcpOptBF1_BF2_BF3 column is to be interpreted as follows:

dhcpOptBF1	Length	Description
2^0 (=0x0000.0000.0000.0001)	0	Pad
2^1 (=0x0000.0000.0000.0002)	4	Subnet Mask
2^2 (=0x0000.0000.0000.0004)	4	Time Offset (deprecated)
2^3 (=0x0000.0000.0000.0008)	4+	Router
2^4 (=0x0000.0000.0000.0010)	4+	Time Server
2^5 (=0x0000.0000.0000.0020)	4+	Name Server
2^6 (=0x0000.0000.0000.0040)	4+	Domain Name Server
2^7 (=0x0000.0000.0000.0080)	4+	Log Server
2^8 (=0x0000.0000.0000.0100)	4+	Quote Server
2^9 (=0x0000.0000.0000.0200)	4+	LPR Server
2^{10} (=0x0000.0000.0000.0400)	4+	Impress Server
2^{11} (=0x0000.0000.0000.0800)	4+	Resource Location Server
2^{12} (=0x0000.0000.0000.1000)	1+	Host Name
2^{13} (=0x0000.0000.0000.2000)	2	Boot File Size
2^{14} (=0x0000.0000.0000.4000)	1+	Merit Dump File
2^{15} (=0x0000.0000.0000.8000)	1+	Domain Name
2^{16} (=0x0000.0000.0001.0000)	4	Swap Server
2^{17} (=0x0000.0000.0002.0000)	1+	Root Path
2^{18} (=0x0000.0000.0004.0000)	1+	Extensions Path
2^{19} (=0x0000.0000.0008.0000)	1	IP Forwarding enable/disable

dhcpOptBF1	Length	Description
2^{20} (=0x0000.0000.0010.0000)	1	Non-local Source Routing enable/disable
2^{21} (=0x0000.0000.0020.0000)	8+	Policy Filter
2^{22} (=0x0000.0000.0040.0000)	2	Maximum Datagram Reassembly Size
2^{23} (=0x0000.0000.0080.0000)	1	Default IP Time-to-live
2^{24} (=0x0000.0000.0100.0000)	4	Path MTU Aging Timeout
2^{25} (=0x0000.0000.0200.0000)	2+	Path MTU Plateau Table
2^{26} (=0x0000.0000.0400.0000)	2	Interface MTU
2^{27} (=0x0000.0000.0800.0000)	1	All Subnets are Local
2^{28} (=0x0000.0000.1000.0000)	4	Broadcast Address
2^{29} (=0x0000.0000.2000.0000)	1	Perform Mask Discovery
2^{30} (=0x0000.0000.4000.0000)	1	Mask supplier
2^{31} (=0x0000.0000.8000.0000)	1	Perform router discovery
2^{32} (=0x0000.0001.0000.0000)	4	Router solicitation address
2^{33} (=0x0000.0002.0000.0000)	8+	Static routing table
2^{34} (=0x0000.0004.0000.0000)	1	Trailer encapsulation
2^{35} (=0x0000.0008.0000.0000)	4	ARP cache timeout
2^{36} (=0x0000.0010.0000.0000)	1	Ethernet encapsulation
2^{37} (=0x0000.0020.0000.0000)	1	Default TCP TTL
2^{38} (=0x0000.0040.0000.0000)	4	TCP keepalive interval
2^{39} (=0x0000.0080.0000.0000)	1	TCP keepalive garbage
2^{40} (=0x0000.0100.0000.0000)	1+	Network Information Service Domain
2^{41} (=0x0000.0200.0000.0000)	4+	Network Information Servers
2^{42} (=0x0000.0400.0000.0000)	4+	NTP servers
2^{43} (=0x0000.0800.0000.0000)	1+	Vendor specific information
2^{44} (=0x0000.1000.0000.0000)	4+	NetBIOS over TCP/IP name server
2^{45} (=0x0000.2000.0000.0000)	4+	NetBIOS over TCP/IP Datagram Distribution Server
2^{46} (=0x0000.4000.0000.0000)	1	NetBIOS over TCP/IP Node Type
2^{47} (=0x0000.8000.0000.0000)	1+	NetBIOS over TCP/IP Scope
2^{48} (=0x0001.0000.0000.0000)	4+	X Window System Font Server
2^{49} (=0x0002.0000.0000.0000)	4+	X Window System Display Manager
2^{50} (=0x0004.0000.0000.0000)	4	Requested IP Address
2^{51} (=0x0008.0000.0000.0000)	4	IP address lease time
2^{52} (=0x0010.0000.0000.0000)	4	Option overload
2^{53} (=0x0020.0000.0000.0000)	4	DHCP message type
2^{54} (=0x0040.0000.0000.0000)	1	Server identifier
2^{55} (=0x0080.0000.0000.0000)	1+	Parameter request list
2^{56} (=0x0100.0000.0000.0000)	1+	Message
2^{57} (=0x0200.0000.0000.0000)	2	Maximum DHCP message size
2^{58} (=0x0400.0000.0000.0000)	4	Renew time value
2^{59} (=0x0800.0000.0000.0000)	4	Rebinding time value
2^{60} (=0x1000.0000.0000.0000)	1+	Class-identifier
2^{61} (=0x2000.0000.0000.0000)	2+	Client-identifier
2^{62} (=0x4000.0000.0000.0000)	1-255	NetWare/IP Domain Name
2^{63} (=0x8000.0000.0000.0000)	1	NetWare/IP information

dhcpOptBF2	Length	Description
2^{64} (=0x0000.0000.0000.0001)	1+	Network Information Service+ Domain
2^{65} (=0x0000.0000.0000.0002)	4+	Network Information Service+ Servers

dhcpOptBF2	Length	Description
2^{66} (=0x0000.0000.0000.0004)	1+	TFTP server name
2^{67} (=0x0000.0000.0000.0008)	1+	Bootfile name
2^{68} (=0x0000.0000.0000.0010)	0+	Mobile IP Home Agen
2^{69} (=0x0000.0000.0000.0020)	4+	Simple Mail Transport Protocol Server
2^{70} (=0x0000.0000.0000.0040)	4+	Post Office Protocol Server
2^{71} (=0x0000.0000.0000.0080)	4+	Network News Transport Protocol Server
2^{72} (=0x0000.0000.0000.0100)	4+	Default World Wide Web Server
2^{73} (=0x0000.0000.0000.0200)	4+	Default Finger Server
2^{74} (=0x0000.0000.0000.0400)	4+	Default Internet Relay Chat Server
2^{75} (=0x0000.0000.0000.0800)	4+	StreetTalk Server
2^{76} (=0x0000.0000.0000.1000)	4+	StreetTalk Directory Assistance Server
2^{77} (=0x0000.0000.0000.2000)	0-255	User Class Information
2^{78} (=0x0000.0000.0000.4000)	0-255	SLP Directory Agent
2^{79} (=0x0000.0000.0000.8000)	0-255	SLP Service Scope
2^{80} (=0x0000.0000.0001.0000)	0	Rapid Commit
2^{81} (=0x0000.0000.0002.0000)	4+	FQDN, Fully Qualified Domain Name
2^{82} (=0x0000.0000.0004.0000)	0-255	Relay Agent Information
2^{83} (=0x0000.0000.0008.0000)	14+	Internet Storage Name Service
2^{84} (=0x0000.0000.0010.0000)	—	—
2^{85} (=0x0000.0000.0020.0000)	8+	—
2^{86} (=0x0000.0000.0040.0000)	2	—
2^{87} (=0x0000.0000.0080.0000)	1	—
2^{88} (=0x0000.0000.0100.0000)	4	—
2^{89} (=0x0000.0000.0200.0000)	2+	—
2^{90} (=0x0000.0000.0400.0000)	2	—
2^{91} (=0x0000.0000.0800.0000)	1	—
2^{92} (=0x0000.0000.1000.0000)	4	—
2^{93} (=0x0000.0000.2000.0000)	1	—
2^{94} (=0x0000.0000.4000.0000)	1	—
2^{95} (=0x0000.0000.8000.0000)	1	—
2^{96} (=0x0000.0001.0000.0000)	—	—
2^{97} (=0x0000.0002.0000.0000)	—	—
2^{98} (=0x0000.0004.0000.0000)	—	—
2^{99} (=0x0000.0008.0000.0000)	—	—
2^{100} (=0x0000.0010.0000.0000)	—	—
2^{101} (=0x0000.0020.0000.0000)	—	—
2^{102} (=0x0000.0040.0000.0000)	—	—
2^{103} (=0x0000.0080.0000.0000)	—	—
2^{104} (=0x0000.0100.0000.0000)	1+	—
2^{105} (=0x0000.0200.0000.0000)	—	—
2^{106} (=0x0000.0400.0000.0000)	—	—
2^{107} (=0x0000.0800.0000.0000)	—	—
2^{108} (=0x0000.1000.0000.0000)	—	—
2^{109} (=0x0000.2000.0000.0000)	—	—
2^{110} (=0x0000.4000.0000.0000)	—	—
2^{111} (=0x0000.8000.0000.0000)	—	—
2^{112} (=0x0001.0000.0000.0000)	—	—
2^{113} (=0x0002.0000.0000.0000)	—	—
2^{114} (=0x0004.0000.0000.0000)	—	—

dhcpOptBF2	Length	Description
2^{115} (=0x0008.0000.0000.0000)	—	—
2^{116} (=0x0010.0000.0000.0000)	—	—
2^{117} (=0x0020.0000.0000.0000)	—	—
2^{118} (=0x0040.0000.0000.0000)	—	—
2^{119} (=0x0080.0000.0000.0000)	—	—
2^{120} (=0x0100.0000.0000.0000)	—	—
2^{121} (=0x0200.0000.0000.0000)	5+	—
2^{122} (=0x0400.0000.0000.0000)	0-255	—
2^{123} (=0x0800.0000.0000.0000)	16	—
2^{124} (=0x1000.0000.0000.0000)	—	—
2^{125} (=0x2000.0000.0000.0000)	—	—
2^{126} (=0x4000.0000.0000.0000)	—	—
2^{127} (=0x8000.0000.0000.0000)	—	—

dhcpOptBF3	Length	Description
2^{128} (=0x0000.0000.0000.0001)	—	TFTP Server IP address
2^{129} (=0x0000.0000.0000.0002)	—	Call Server IP addres
2^{130} (=0x0000.0000.0000.0004)	—	Discrimination string
2^{131} (=0x0000.0000.0000.0008)	—	Remote statistics server IP address
2^{132} (=0x0000.0000.0000.0010)	—	802.1P VLAN ID
2^{133} (=0x0000.0000.0000.0020)	—	802.1Q L2 Priority
2^{134} (=0x0000.0000.0000.0040)	—	Diffserv Code Point
2^{135} (=0x0000.0000.0000.0080)	—	HTTP Proxy for phone-specific applications
2^{136} (=0x0000.0000.0000.0100)	4+	PANA Authentication Agent
2^{137} (=0x0000.0000.0000.0200)	0-255	LoST Server
2^{138} (=0x0000.0000.0000.0400)	—	CAPWAP Access Controller addresses
2^{139} (=0x0000.0000.0000.0800)	—	OPTION-IPv4_Address-MoS
2^{140} (=0x0000.0000.0000.1000)	—	OPTION-IPv4_FQDN-MoS
2^{141} (=0x0000.0000.0000.2000)	2+	SIP UA Configuration Service Domains
2^{142} (=0x0000.0000.0000.4000)	—	OPTION-IPv4_Address-ANDSF
2^{143} (=0x0000.0000.0000.8000)	—	OPTION-IPv6_Address-ANDSF
2^{144} (=0x0000.0000.0001.0000)	—	—
2^{145} (=0x0000.0000.0002.0000)	—	—
2^{146} (=0x0000.0000.0004.0000)	—	—
2^{147} (=0x0000.0000.0008.0000)	—	—
2^{148} (=0x0000.0000.0010.0000)	—	—
2^{149} (=0x0000.0000.0020.0000)	—	—
2^{150} (=0x0000.0000.0040.0000)	—	TFTP server address or Etherboot-GRUB configuration path name
2^{151} (=0x0000.0000.0080.0000)	—	status-code
2^{152} (=0x0000.0000.0100.0000)	—	base-time
2^{153} (=0x0000.0000.0200.0000)	—	start-time-of-state
2^{154} (=0x0000.0000.0400.0000)	—	query-start-time
2^{155} (=0x0000.0000.0800.0000)	—	query-end-time
2^{156} (=0x0000.0000.1000.0000)	—	dhcp-state
2^{157} (=0x0000.0000.2000.0000)	—	data-source
2^{158} (=0x0000.0000.4000.0000)	—	—
2^{159} (=0x0000.0000.8000.0000)	—	—
2^{160} (=0x0000.0001.0000.0000)	—	—

dhcpOptBF3	Length	Description
2^{161} (=0x0000.0002.0000.0000)	—	—
2^{162} (=0x0000.0004.0000.0000)	—	—
2^{163} (=0x0000.0008.0000.0000)	—	—
2^{164} (=0x0000.0010.0000.0000)	—	—
2^{165} (=0x0000.0020.0000.0000)	—	—
2^{166} (=0x0000.0040.0000.0000)	—	—
2^{167} (=0x0000.0080.0000.0000)	—	—
2^{168} (=0x0000.0100.0000.0000)	—	—
2^{169} (=0x0000.0200.0000.0000)	—	—
2^{170} (=0x0000.0400.0000.0000)	—	—
2^{171} (=0x0000.0800.0000.0000)	—	—
2^{172} (=0x0000.1000.0000.0000)	—	—
2^{173} (=0x0000.2000.0000.0000)	—	—
2^{174} (=0x0000.4000.0000.0000)	—	—
2^{175} (=0x0000.8000.0000.0000)	—	Etherboot
2^{176} (=0x0001.0000.0000.0000)	—	IP Telephone
2^{177} (=0x0002.0000.0000.0000)	—	Etherboot, PacketCable and CableHome
2^{178} (=0x0004.0000.0000.0000)	—	—
2^{179} (=0x0008.0000.0000.0000)	—	—
2^{180} (=0x0010.0000.0000.0000)	—	—
2^{181} (=0x0020.0000.0000.0000)	—	—
2^{182} (=0x0040.0000.0000.0000)	—	—
2^{183} (=0x0080.0000.0000.0000)	—	—
2^{184} (=0x0100.0000.0000.0000)	—	—
2^{185} (=0x0200.0000.0000.0000)	—	—
2^{186} (=0x0400.0000.0000.0000)	—	—
2^{187} (=0x0800.0000.0000.0000)	—	—
2^{188} (=0x1000.0000.0000.0000)	—	—
2^{189} (=0x2000.0000.0000.0000)	—	—
2^{190} (=0x4000.0000.0000.0000)	—	—
2^{191} (=0x8000.0000.0000.0000)	—	—

10.4 Packet File Output

In packet mode (`-s` option), the dhcpDecode plugin outputs the following columns:

Column	Type	Description
dhcpMType	U8	Message type
dhcpHops	U8	Number of hops
dhcpTransID	U16	Transaction Identifier
dhcpLFlow	U16	Linked flow

10.5 Plugin Report Output

The number of DHCP packets of each type (Section 10.3.2) is reported.

10.6 TODO

- DHCPv6

10.7 References

- RFC2131: Dynamic Host Configuration Protocol
- RFC2132: DHCP Options and BOOTP Vendor Extensions

11 dnsDecode

11.1 Description

This plugin produces DNS header and content information encountered during the lifetime of a flow. The idea is to identify DNS header and payload features using flow parameters in order to extract information about applications or users. The DNS plugin requires no dependencies and produces only output to the flow file. User defined compiler switches in *dnsDecode.h, malsite.h* produce optimized code for the specific application.

11.2 Configuration Flags

The flow based output and the extracted information can be controlled by switches and constants listed in the table below. The most important one is `DNS_MODE` which controls the amount of information in the flow file. `DNS_AGGR` controls the aggregation of duplicate names and values. The last three limit the amount of memory allocated for flow based DNS record storage. The default values revealed reasonable performance in practise.

Name	Default	Description	Flags
DNS_MODE	4	0: Only aggregated header count info 1: +REQ records 2: +ANS records 3: +AUX records 4: +ADD records	
DNS_HEXON	1	0: Hex Output flags off, 1: Hex output flags on	
DNS_REQA	0	0: full vectors, 1: Aggregate request records	
DNS_ANSA	0	0: full vectors, 1: Aggregate answer records	
DNS_QRECMAX	15	Max # of query records / flow	
DNS_ARECMAX	20	Max # of answer records / flow	
MAL_TEST	0	1: activate blacklist malware test mode (IPv4 only)	
MAL_TYPE	0	1: Type string; 0: Code	
The following additional flag is available in `malsite.h`:			
MAL_DOMAIN	1	0: malsite ip address labeling mode 1: malsite domain labeling mode	

11.3 Flow File Output

The default settings will result in 11 tab separated columns in the flow file where the items in column 6-11 are sequences of strings containing DNS record name, address entries and specific DNS entry information such as Type or TTL separated by semicolons. The idea is that the array elements of strings of the different columns correspond to each other so that easy script based post processing is possible. The different output modes controlled by `DNS_MODE` provide an incremental method from a high speed compressed representation to a full human readable representation.

Column	Type	Description	Flags
dnsStat	H16	Status, warnings and errors	
dnsHdriOPField	H16	Header field of last packet in flow	
DnsStat_	H8_	Aggregated header status,	

Column	Type	Description	Flags
OpC_	H16_	opcode and	
RetC	H16	return code	
dnsCntQu_	R:U16_	# of question records	
Asw_	U16_	# answer records	
Aux_	U16_	# of auxiliary records	
Add	U16	# additional records	
dnsAAAqF	F	DDOS DNS AAA / Query factor	
dnsTypeBF3_BF2_BF1_BF0	H8_H16_H16_H64	Type bitfields	DNS_MODE > 0
dnsQname	RS	Query Name records	DNS_MODE > 1
dnsMalType	RS	Domain Malware Type String	MAL_TEST=1 && MAL_TYPE=1 && MAL_DOMAIN=1
dnsMalCode	RH32	Domain Malware code	MAL_TEST=1 && MAL_TYPE=0 && MAL_DOMAIN=1
dnsAname	RS	Answer Name records	
dnsAPname	RS	Name CNAME entries	
dns4Aaddress	RIP4	Address entries IPv4	
dns6Aaddress	RIP6	Address entries IPv6	
dnsIPMalCode	RH32	IP Malware code	MAL_TEST=1 && MAL_DOMAIN=0
dnsAType	RU16	Answer record Type entries	
dnsAClass	RU16	Answer record Class entries	
dnsATTL	RU32	Answer record TTL entries	
dnsMXpref	RU16	MX record preference entries	
dnsSRVprio	RU16	SRV record priority entries	
dnsSRVwgt	RU16	SRV record weight entries	
dnsOptStat	RU32	option status	
dnsOptCodeOwn	RU16	option code owner	

11.3.1 dnsStat

The DNS status bit field listed below provides an efficient method to post process flow data files in order to detect incidents during flow processing.

dnsStat	Type	Description
2^0 (=0x0001)	DNS_PRTDT	DNS ports detected
2^1 (=0x0002)	DNS_NBIOS	NetBios DNS
2^2 (=0x0004)	DNS_FRAGA	DNS TCP aggregated fragmented content
2^3 (=0x0008)	DNS_FRAGS	DNS TCP fragmented content state
2^4 (=0x0010)	DNS_FTRUNC	Warning: Name truncated
2^5 (=0x0020)	DNS_ANY	Warning: ANY: Zone all from a domain or cached server
2^6 (=0x0040)	DNS_IZTRANS	Warning: Incremental DNS zone transfer detected
2^7 (=0x0080)	DNS_ZTRANS	Warning: DNS zone transfer detected

dnsStat	Type	Description
2^8 (=0x0100)	DNS_WRNULN	Warning: DNS UDP Length exceeded
2^9 (=0x0200)	DNS_WRNIGN	Warning: following Records ignored
2^{10} (=0x0400)	DNS_WRNDEX	Warning: Max DNS name records exceeded
2^{11} (=0x0800)	DNS_WRNAEX	Warning: Max address records exceeded
2^{12} (=0x1000)	DNS_ERRLEN	Error: DNS record length error
2^{13} (=0x2000)	DNS_ERRPTR	Error: Wrong DNS PTR detected
2^{14} (=0x4000)	DNS_WRNMLN	Warning: DNS length undercut
2^{15} (=0x8000)	DNS_ERRCRPT	Error: UDP/TCP DNS Header corrupt or TCP packets missing

11.3.2 dnsHdriOPField

From the 16 Bit DNS header the QR Bit and Bit five to nine are extracted and mapped in their correct sequence into a byte as indicated below. It provides for a normal single packet exchange flow an accurate status of the DNS transfer. For a multiple packet exchange only the last packet is mapped into the variable. In that case the aggregated header state flags should be considered.

QR	Opcode	AA	TC	RD	RA	Z	AD	CD	Rcode
1	0000	1	0	1	1	1	0	0	0000

11.3.3 dnsHStat_OpC_RetC

For multi-packet DNS flows e.g. via TCP the aggregated header state bit field describes the status of all packets in a flow. Thus, flows with certain client and server states can be easily identified and extracted during post-processing.

dnsHStat	Short	Description
2^7 (=0x01)	CD	Checking Disabled
2^6 (=0x02)	AD	Authenticated Data
2^5 (=0x04)	Z	Zero
2^4 (=0x08)	RA	Recursion Available
2^3 (=0x10)	RD	Recursion Desired
2^2 (=0x20)	TC	Truncated
2^1 (=0x40)	AA	Authoritative Answer
2^0 (=0x80)	QR	Query / Response

The four bit OpCode field of the DNS header is mapped via [2^{Opcode}] and an OR into a 16 Bit field. Thus, the client can be monitored or anomalies easily identified. E.g. appearance of reserved bits might be an indication for a covert channel or malware operation.

dnsOpC	Description
2^0 (=0x0001)	QUERY, Standard query
2^1 (=0x0002)	IQUERY, Inverse query
2^2 (=0x0004)	STATUS, Server status request
2^3 (=0x0008)	—
2^4 (=0x0010)	Notify

dnsOpC	Description
2^4 (=0x0020)	Update
2^5 (=0x0040)	reserved
2^6 (=0x0080)	reserved
2^8 (=0x0100)	reserved
2^9 (=0x0200)	reserved
2^{10} (=0x0400)	reserved
2^{11} (=0x0800)	reserved
2^{12} (=0x1000)	reserved
2^{13} (=0x2000)	reserved
2^{14} (=0x4000)	reserved
2^{15} (=0x8000)	reserved

The four bit RCode field of the DNS header is mapped via [2^{Rcode}] and an OR into a 16 Bit field. It provides valuable information about success of DNS queries and therefore facilitates the detection of failures, misconfigurations and malicious operations.

dnsRetC	Short	Description
2^0 (=0x0001)	No error	Request completed successfully
2^1 (=0x0002)	Format error	Name server unable to interpret query
2^2 (=0x0004)	Server failure	Name server unable to process query due to problem with name server
2^3 (=0x0008)	Name Error	Authoritative name server only: Domain name in query does not exist
2^4 (=0x0010)	Not Implemented	Name server does not support requested kind of query.
2^4 (=0x0020)	Refused	Name server refuses to perform the specified operation for policy reasons.
2^5 (=0x0040)	YXDomain	Name Exists when it should not
2^6 (=0x0080)	YXRRSet	RR Set Exists when it should not
2^8 (=0x0100)	NXRRSet	RR Set that should exist does not
2^9 (=0x0200)	NotAuth	Server Not Authoritative for zone
2^{10} (=0x0400)	NotZone	Name not contained in zone
2^{11} (=0x0800)	—	—
2^{12} (=0x1000)	—	—
2^{13} (=0x2000)	—	—
2^{14} (=0x4000)	—	—
2^{15} (=0x8000)	—	—

11.3.4 dnsTypeBF3_BF2_BF1_BF0

The 16 bit Type Code field is extracted from each DNS record and mapped via [2^{Typecode}] into a 64 Bit fields. Gaps are avoided by additional higher bitfields defining higher codes.

dnsTypeBF3	Short	Description
2^0 (=0x01)	TA	DNSSEC Trust Authorities
2^1 (=0x02)	DLV	DNSSEC Lookaside Validation
2^2 (=0x04)	—	—
2^3 (=0x08)	—	—

dnsTypeBF3	Short	Description
2^4 (=0x10)	—	—
2^5 (=0x20)	—	—
2^6 (=0x40)	—	—
2^7 (=0x80)	—	—

dnsTypeBF2	Short	Description
2^0 (=0x0001)	TKEY	Transaction Key
2^1 (=0x0002)	TSIG	Transaction Signature
2^2 (=0x0004)	IXFR	Incremental transfer
2^3 (=0x0008)	AXFR	Transfer of an entire zone
2^4 (=0x0010)	MAILB	Mailbox-related RRs (MB, MG or MR)
2^5 (=0x0020)	MAILA	Mail agent RRs (OBSOLETE - see MX)
2^6 (=0x0040)	ZONEALL	Request for all records the server/cache has available
2^7 (=0x0080)	URI	URI
2^8 (=0x0100)	CAA	Certification Authority Restriction
2^9 (=0x0200)	—	—
2^{10} (=0x0400)	—	—
2^{11} (=0x0800)	—	—
2^{12} (=0x1000)	—	—
2^{13} (=0x2000)	—	—
2^{14} (=0x4000)	—	—
2^{15} (=0x8000)	—	—

dnsTypeBF1	Short	Description
2^0 (=0x0001)	SPF	
2^1 (=0x0002)	UINFO	
2^2 (=0x0004)	UID	
2^3 (=0x0008)	GID	
2^4 (=0x0010)	UNSPEC	
2^4 (=0x0020)	NID	
2^5 (=0x0040)	L32	
2^6 (=0x0080)	L64	
2^8 (=0x0100)	LP	
2^9 (=0x0200)	EUI48	EUI-48 address
2^{10} (=0x0400)	EUI64	EUI-48 address
2^{11} (=0x0800)	—	—
2^{12} (=0x1000)	—	—
2^{13} (=0x2000)	—	—
2^{14} (=0x4000)	—	—
2^{15} (=0x8000)	—	—

dnsTypeBF0	Short	Description
2^0 (=0x0000.0000.0000.0001)	—	—
2^1 (=0x0000.0000.0000.0002)	A	IPv4 address
2^2 (=0x0000.0000.0000.0004)	NS	Authoritative name server
2^3 (=0x0000.0000.0000.0008)	MD	Mail destination. Obsolete use MX instead
2^4 (=0x0000.0000.0000.0010)	MF	Mail forwarder. Obsolete use MX instead
2^5 (=0x0000.0000.0000.0020)	CNAME	Canonical name for an alias
2^6 (=0x0000.0000.0000.0040)	SOA	Marks the start of a zone of authority
2^7 (=0x0000.0000.0000.0080)	MB	Mailbox domain name
2^8 (=0x0000.0000.0000.0100)	MG	Mail group member
2^9 (=0x0000.0000.0000.0200)	MR	Mail rename domain name
2^{10} (=0x0000.0000.0000.0400)	NULL	Null resource record
2^{11} (=0x0000.0000.0000.0800)	WKS	Well known service description
2^{12} (=0x0000.0000.0000.1000)	PTR	Domain name pointer
2^{13} (=0x0000.0000.0000.2000)	HINFO	Host information
2^{14} (=0x0000.0000.0000.4000)	MINFO	Mailbox or mail list information
2^{15} (=0x0000.0000.0000.8000)	MX	Mail exchange
2^{16} (=0x0000.0000.0001.0000)	TXT	Text strings
2^{17} (=0x0000.0000.0002.0000)	—	Responsible Person
2^{18} (=0x0000.0000.0004.0000)	AFSDB	AFS Data Base location
2^{19} (=0x0000.0000.0008.0000)	X25	X.25 PSDN address
2^{20} (=0x0000.0000.0010.0000)	ISDN	ISDN address
2^{21} (=0x0000.0000.0020.0000)	RT	Route Through
2^{22} (=0x0000.0000.0040.0000)	NSAP	NSAP address. NSAP style A record
2^{23} (=0x0000.0000.0080.0000)	NSAP-PTR	—
2^{24} (=0x0000.0000.0100.0000)	SIG	Security signature
2^{25} (=0x0000.0000.0200.0000)	KEY	Security key
2^{26} (=0x0000.0000.0400.0000)	PX	X.400 mail mapping information
2^{27} (=0x0000.0000.0800.0000)	GPOS	Geographical Position
2^{28} (=0x0000.0000.1000.0000)	AAAA	IPv6 Address
2^{29} (=0x0000.0000.2000.0000)	LOC	Location Information
2^{30} (=0x0000.0000.4000.0000)	NXT	Next Domain (obsolete)
2^{31} (=0x0000.0000.8000.0000)	EID	Endpoint Identifier
2^{32} (=0x0000.0001.0000.0000)	NIMLOC/NB	Nimrod Locator / NetBIOS general Name Service
2^{33} (=0x0000.0002.0000.0000)	SRV/NBSTAT	Server Selection / NetBIOS NODE STATUS
2^{34} (=0x0000.0004.0000.0000)	ATMA	ATM Address
2^{35} (=0x0000.0008.0000.0000)	NAPTR	Naming Authority Pointer
2^{36} (=0x0000.0010.0000.0000)	KX	Key Exchanger
2^{37} (=0x0000.0020.0000.0000)	CERT	—
2^{38} (=0x0000.0040.0000.0000)	A6	A6 (OBSOLETE - use AAAA)
2^{39} (=0x0000.0080.0000.0000)	DNAME	—
2^{40} (=0x0000.0100.0000.0000)	SINK	—
2^{41} (=0x0000.0200.0000.0000)	OPT	—
2^{42} (=0x0000.0400.0000.0000)	APL	—
2^{43} (=0x0000.0800.0000.0000)	DS	Delegation Signer
2^{44} (=0x0000.1000.0000.0000)	SSHFP	SSH Key Fingerprint

dnsTypeBF0	Short	Description
2^{45} (=0x0000.2000.0000.0000)	IPSECKEY	—
2^{46} (=0x0000.4000.0000.0000)	RRSIG	—
2^{47} (=0x0000.8000.0000.0000)	NSEC	NextSECure
2^{48} (=0x0001.0000.0000.0000)	DNSKEY	—
2^{49} (=0x0002.0000.0000.0000)	DHCID	DHCP identifier
2^{50} (=0x0004.0000.0000.0000)	NSEC3	—
2^{51} (=0x0008.0000.0000.0000)	NSEC3PARAM	—
2^{52} (=0x0010.0000.0000.0000)	TLSA	—
2^{53} (=0x0020.0000.0000.0000)	SMIMEA	S/MIME cert association
2^{54} (=0x0040.0000.0000.0000)	—	
2^{55} (=0x0080.0000.0000.0000)	HIP	Host Identity Protocol
2^{56} (=0x0100.0000.0000.0000)	NINFO	—
2^{57} (=0x0200.0000.0000.0000)	RKEY	—
2^{58} (=0x0400.0000.0000.0000)	TALINK	Trust Anchor LINK
2^{59} (=0x0800.0000.0000.0000)	CDS	Child DS
2^{60} (=0x1000.0000.0000.0000)	CDNSKEY	DNSKEY(s) the Child wants reflected in DS
2^{61} (=0x2000.0000.0000.0000)	OPENPGPKEY	OpenPGP Key
2^{62} (=0x4000.0000.0000.0000)	CSYNC	Child-To-Parent Synchronization
2^{63} (=0x8000.0000.0000.0000)	—	

11.4 Plugin Report Output

The following information is reported:

- Number of DNS IPv4/6 packets
- Number of DNS IPv4/6 Q,R packets
- Aggregated status flags (dnsStat)
- Number of alarms (MAL_TEST)

11.5 Example Output

The idea is that the string and integer array elements of question, answer, TTL and Type record entries match by column index so that easy script based mapping and post processing is possible. A sample output is shown below. Especially when large records are present the same name is printed several times which might degrade the readability. Therefore, a next version will have a multiple Aname suppressor switch, which should be off for script based post-processing.

Query name	Answer name	Answer address	TTL	Type
www.macromedia.com;	www.macromedia.com;www-mm.wip4.adobe.com	0.0.0.0;8.118.124.64	2787;4	5;1

11.6 TODO

- Compressed mode for DNS records

12 entropy

12.1 Description

The entropy plugin calculates the entropy of the snapped IP payload distribution. The calculation of the entropy demands a number elements equal to the SQR(alphabet) = 16 in the default case. The size of the alphabet is variable. By default, one byte = 256 characters. Two other key parameters, a binary and text based ratio, in combination with the entropy serve as input for AI for content and application classification. The character and binary ratio denote the degree of text or binary content respectively.

The entropy plugin operates in two modes:

- entropy payload
- entropy payload + time series

and for production purposes by default deactivated. The parameter `ENT_MAXPBIN` controls the size of the alphabet and `ENT_ALPHA_D` the output of the payload character distribution per flow.

12.1.1 Entropy Time Series (Experimental)

The reason for this flow file addition is the exploration of entropy chunks calculated over the whole payload as a series.

12.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
`ENT_THRES`	1	calc entropy only if number of payload bytes >
`ENT_ALPHA_D`	0	1: print Alphabet distribution in flow file
`ENT_D_OFFSET`	0	start of entropy calc in payload
The following flags are experimental for the MAC anomaly detection end report:		
`ENT_FLOW`	0	global flow entropy: 1: entropy, 0 output; 2: + distribution
`ENT_NTUPLE`	55	

12.3 Flow File Output

The entropy plugin outputs the following columns:

Column	Type	Description	Flags
`PyldEntropy`	F	Payload entropy: no entropy calculated:-1.0	
`PyldChRatio`	F	Payload Character ratio	
`PyldBinRatio`	F	Payload Binary ratio	
`Pyldlen`	U32	Payload length	`ENT_ALPHA_D=1`
`PyldHisto`	RU32	Payload histogram	`ENT_ALPHA_D=1`

13 findexer

This plugin produces a binary index mapping each flow index to its packets positions in the input pcaps. The goal of this plugin is to be able to quickly extract flows from a big pcap without having to re-process it completely. The `fextractor` tool can be used to extract flows from the pcaps using the generated index.

13.1 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
`FINDEXER_SPLIT`	1	Whether (1) or not (0) to split the findexer file with t2 `-W` option

13.2 fextractor

The `fextractor` tool can be used to extract flows using the generated `_flows.xer` index.

```
Usage: fextractor -r INPUT[:start][,end] (-w OUTPUT | -n) [OPTIONS]... \
            [[DIR@]FLOWINDEX[:start][,end]]...

Extract the flows FLOWINDEX using the _flows.xer INTPUT generated by Tranalyzer2 findexer plugin.
Alternatively use a list of findexer files generated by Tranalyzer2 -W option from index start
to end. The extracted flows are written to the OUTPUT pcap.

An optional packet range can be provided on each command line FLOWINDEX to only extract packets
in the range [start, end] of this flow. If start or end are ommitted, they are replaced by,
respectively, the first and the last available packets in the flow. The FLOWINDEX can also
optionally be prefixed with a direction A or B, by default both directions are extracted.

OPTIONS:
  -r INPUT[:start][,end]
            either read packet indexes from a single _flows.xer file named INPUT
            or read packet indexes from multiple _flows.xer files prefixed by INPUT
            and with suffix in range [start, end]. If start or end are ommitted,
            they are replaced by, respectively, first and last available PCAP.
  -w OUTPUT write packets to pcap file OUTPUT
            OUTPUT "-" means that the PCAP is written to stdout.
  -f        overwrite OUTPUT if it already exists
  -n        print oldest PCAP still available, its first packet timestamp and exit
  -h        print this help message
  -i FILE   read flow indexes from FILE. FILE can either be in _flows.txt format
            (flow index in 2nd tab-separated column), or have one flow index per line.
            FILE "-" means that flows are read from stdin.
  -b        by default when FILE is in _flows.txt format, only directions present in
            it are extracted, this option force both directions to be extracted even if
            only the A or B direction is present in the flow file.
  -s N      skip the first N PCAPs
  -p DIR    search pcaps in DIR
            should only be set if pcaps were moved since Tranalyzer2 was run
```

Example to extract flow 42, 123 and 1337 to the `output.pcap` file:

```
fextractor -r ~/t2_output/dmp1_flows.xer -w output.pcap 42 123 1337
```

13.3 Example scenario

We want to extract all the flows whose source or destination are in China, to look at them in Wireshark.

First, we run tranalyzer with at least the `findexer`, `basicFlow` and `txtSink` plugins. The findexer plugin will generate a `_flows.xer` index file which keeps a list of packets positions in the original PCAP for each flow.

```
[user@machine]$ tranalyzer -r capture01.pcap -w t2_output/capture01
```

We now use the `srcIPCC` and `dstIPCC` columns to filter flows with IPs in China.

```
[user@machine]$ grep IPCC t2_output/capture01_headers.txt
9       SC:N    srcIPCC Source IP country code
12      SC:N    dstIPCC Destination IP country code
```

The country code are in the 9 and 12 columns. The flows to extract can directly be piped to the fextractor which then pipe the extracted PCAP to Wireshark.

```
[user@machine]$ awk -F"\t" '$9 == "cn" || $12 == "cn"' t2_output/capture01_flows.txt | \
        fextractor -i - -r t2_output/capture01_flows.xer -w - | wireshark -k -i -
```

By using `tawk` we don't even need to look at the column numbers in the header file, we can directly extract the flows of interest using the column names. `tawk` also provides a `-k` option which takes care of extracting the flows and opening them in Wireshark.

```
[user@machine]$ tawk -k '$srcIPCC == "cn" || $dstIPCC == "cn"' t2_output/capture01_flows.txt
```

13.4 Additional Output (findexer v2)

A binary index with suffix `_flows.xer` is generated. This file is composed of the following sections in any order (except the findexer header which is always at the beginning of the file). All numbers are written in little endian.

findexer header

```
struct findexer_header {
    uint64_t magic;          // 0x32455845444e4946 = FINDEXE2
    uint32_t pcapCount;      // number of input pcaps provided to tranalyzer
                             // 1 if -r or number of lines in -R file
    uin64_t firstPcapHeader; // offset of the fist pcap headers (see next section)
                             // in the _flows.xer file
};
```

pcap header

```
struct pcap_header {
    uint64_t nextPcapHeader;  // offset of the next pcap header
                              // in the _flows.xer file
    uint64_t flowCount;       // number of flows in this pcap
    uint64_t firstFlowHeader; // offset of the first flow header (see next section)
                              // of this pcap in the _flows.xer file
    uint16_t pathLength;      // length of the path string
    char* pcapPath;           // path string (NOT null terminated)
};
```

flow header

```
struct flow_header {
    uint64_t nextFlowHeader; // offset of the next flow header
                             // in the _flows.xer file
    uint64_t flowIndex;      // Tranalyzer flow index (2nd column in flow file)
    uint8_t flags;           // flow flags (see next section)
    uint64_t packetCount;    // number of packets in this flow
#FOREACH packet in the flow
    uin64t_t offset;         // offset in the pcap where to find the packet
#ENDFOREACH
};
```

flow flags

flags	Description
2^0 (=0x01)	This is a B flow.
2^1 (=0x02)	This is the first XER file in which this flow appears.
2^2 (=0x04)	This is the last XER file in which this flow appears.
2^3 (=0x08)	and all higher values: reserved for future use.

13.5 Limitations

- PcapNg format is not supported (packet offsets in the pcap cannot be computed because of the additional block structures). PcapNg can however be converted in standard Pcap using the following command:

  ```
  editcap -F pcap input.pcapng output.pcap
  ```

- The findexer file cannot be generated when a BPF is used. With a BPF, not all packets are processed by Tranalyzer2 which makes it impossible to compute packets offsets in a PCAP.

13.6 Old format (findexer v1)

findexer header

```
struct findexer_header {
    uint64_t magic;           // 0x52455845444e4946 = FINDEXER
```

```
    uint32_t pcapCount;       // number of input pcaps provided to tranalyzer
                              // 1 if -r or number of lines in -R file
#FOREACH pcap
    uin64_t pcapHeaderOffset; // offset of the per pcap headers (see next section)
                              // in the _flows.xer file
#ENDFOREACH
};
```

pcap header

```
struct pcap_header {
    uint16_t pathLength;    // length of the path string
    char* pcapPath;         // path string (NOT null terminated)
    uint64_t flowCount;     // number of flows in this pcap
#FOREACH flow
    uint64_t flowIndex;     // Tranalyzer flow index (2nd column in flow file)
    uint64_t packetCount;   // number of packets in this flow
    uint64_t packetsOffset; // offset in the _flows.xer file where this flow packet
                            // offsets in the pcap (see next section) are located
#ENDFOREACH
};
```

packet offsets

```
#FOREACH packet in the flow
    uin64_t offset; // offset in the pcap where to find the packet
#ENDFOREACH
```

To extract flow `123`, the following steps are followed:

- open the `_flows.xer` file and check it has the right `magic` value
- for each pcap in `pcapCount`
  - read the pcap header located at `pcapHeaderOffset` in the `_find.xer` file.
  - for each flow in `flowCount`
    - if `flowIndex == 123`: read `packetCount` offsets at position `packetsOffset` in the `_flows.xer` file and extract packets located at these offsets in the pcap at `pcapPath`

14 fnameLabel

14.1 Description

The fnameLabel plugin tags every flow with the name of the file or interface from which the flow originates.

14.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description	Flags
FNL_IDX	1	Use the 'FNL_IDX' letter of the filename as label	

14.3 Flow File Output

The fnameLabel plugin outputs the following columns:

Column	Type	Description
fnLabel	U8	FNL_IDX letter of the filename/interface
fnHash	U64	Hash of the filename/interface
fname	S	Filename

15 ftpDecode

15.1 Description

The ftpDecode plugin analyses FTP traffic. User defined compiler switches are in *ftpDecode.h*.

15.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
FTP_SAVE	0	Save content to FTP_F_PATH
BITFIELD	0	Bitfield coding of FTP commands
FTP_MXNMUN	10	maximal USER name length
FTP_MXNMPN	10	maximal PW length
FTP_MXNMLN	50	maximal name length
FTP_MAXCPFI	10	Maximal number of parent findex
MAXUNM	5	maximal number of users
MAXPNM	5	maximal number of passwords
MAXCNM	20	maximal number of parameters
FTP_F_PATH	"/tmp/FTPFILES/"	Path for extracted content

The plugin identifies the client ftp flows automatically and links them via the ftpCDFindex, identifiying the findex of the associated flows.

15.3 Flow File Output

The ftpDecode plugin outputs the following columns:

Column	Type	Description	Flags
ftpStat	H8	Status bit field	
ftpCBF	H64	Command bit field	BITFIELD=1
ftpCDFindex	RU64	Command/data findex link	
ftpCC	RSC	FTP Command Codes	
ftpRC	RU16	FTP Response Codes	
ftpUsrNum	U8	number of FTP users	
ftpPwNum	U8	number of FTP passwords	
ftpCNum	U8	number of FTP parameters	
ftpUsr	RS	FTP users	
ftpPw	RS	FTP passwords	
ftpC	RS	FTP content	

15.3.1 ftpStat

The ftpStat column describes the errors encountered during the flow lifetime:

ftpStat	Name	Description
2^0 (=0x01)	FTP control port found	
2^1 (=0x02)	FTP passive parent flow	
2^2 (=0x04)	FTP passive parent flow write finished	
2^3 (=0x08)	FTP active parent flow	
2^4 (=0x10)	FTP hash map full	
2^5 (=0x20)	File error	
2^6 (=0x40)	Data flow not detected	
2^7 (=0x80)	Array overflow	

15.3.2 ftpCBF

The `ftpCBF` column is to be interpreted as follows:

ftpCBF	Description
2^0 (=0x0000.0000.0000.0001)	ABOR
2^1 (=0x0000.0000.0000.0002)	ACCT
2^2 (=0x0000.0000.0000.0004)	ADAT
2^3 (=0x0000.0000.0000.0008)	ALLO
2^4 (=0x0000.0000.0000.0010)	APPE
2^5 (=0x0000.0000.0000.0020)	AUTH
2^6 (=0x0000.0000.0000.0040)	CCC
2^7 (=0x0000.0000.0000.0080)	CDUP
2^8 (=0x0000.0000.0000.0100)	CONF
2^9 (=0x0000.0000.0000.0200)	CWD
2^{10} (=0x0000.0000.0000.0400)	DELE
2^{11} (=0x0000.0000.0000.0800)	ENC
2^{12} (=0x0000.0000.0000.1000)	EPRT
2^{13} (=0x0000.0000.0000.2000)	EPSV
2^{14} (=0x0000.0000.0000.4000)	FEAT
2^{15} (=0x0000.0000.0000.8000)	HELP
2^{16} (=0x0000.0000.0001.0000)	LANG
2^{17} (=0x0000.0000.0002.0000)	LIST
2^{18} (=0x0000.0000.0004.0000)	LPRT
2^{19} (=0x0000.0000.0008.0000)	LPSV
2^{20} (=0x0000.0000.0010.0000)	MDTM
2^{21} (=0x0000.0000.0020.0000)	MIC
2^{22} (=0x0000.0000.0040.0000)	MKD
2^{23} (=0x0000.0000.0080.0000)	MLSD
2^{24} (=0x0000.0000.0100.0000)	MLST
2^{25} (=0x0000.0000.0200.0000)	MODE
2^{26} (=0x0000.0000.0400.0000)	NLST
2^{27} (=0x0000.0000.0800.0000)	NOOP
2^{28} (=0x0000.0000.1000.0000)	OPTS
2^{29} (=0x0000.0000.2000.0000)	PASS
2^{30} (=0x0000.0000.4000.0000)	PASV

ftpCBF	Description
2^{31} (=0x0000.0000.8000.0000)	PBSZ
2^{32} (=0x0000.0001.0000.0000)	PORT
2^{33} (=0x0000.0002.0000.0000)	PROT
2^{34} (=0x0000.0004.0000.0000)	PWD
2^{35} (=0x0000.0008.0000.0000)	QUIT
2^{36} (=0x0000.0010.0000.0000)	REIN
2^{37} (=0x0000.0020.0000.0000)	REST
2^{38} (=0x0000.0040.0000.0000)	RETR
2^{39} (=0x0000.0080.0000.0000)	RMD
2^{40} (=0x0000.0100.0000.0000)	RNFR
2^{41} (=0x0000.0200.0000.0000)	RNTO
2^{42} (=0x0000.0400.0000.0000)	SITE
2^{43} (=0x0000.0800.0000.0000)	SIZE
2^{44} (=0x0000.1000.0000.0000)	SMNT
2^{45} (=0x0000.2000.0000.0000)	STAT
2^{46} (=0x0000.4000.0000.0000)	STOR
2^{47} (=0x0000.8000.0000.0000)	STOU
2^{48} (=0x0001.0000.0000.0000)	STRU
2^{49} (=0x0002.0000.0000.0000)	SYST
2^{50} (=0x0004.0000.0000.0000)	TYPE
2^{51} (=0x0008.0000.0000.0000)	USER
2^{52} (=0x0010.0000.0000.0000)	XCUP
2^{53} (=0x0020.0000.0000.0000)	XMKD
2^{54} (=0x0040.0000.0000.0000)	XPWD
2^{55} (=0x0080.0000.0000.0000)	XRCP
2^{56} (=0x0100.0000.0000.0000)	XRMD
2^{57} (=0x0200.0000.0000.0000)	XRSQ
2^{58} (=0x0400.0000.0000.0000)	XSEM
2^{59} (=0x0800.0000.0000.0000)	XSEN
2^{60} (=0x1000.0000.0000.0000)	CLNT

16 geoip

16.1 Description

This plugin outputs the geographic location of IP addresses.

16.2 Dependencies

This product includes GeoLite2 data created by MaxMind, available from http://www.maxmind.com.
Legacy databases (GeoLiteCity.data.gz and GeoLiteCityv6.dat.gz) require *libgeoip*, while GeoLite2 requires *libmaxminddb*.

Ubuntu: `sudo apt-get install libgeoip-dev libmaxminddb-dev`

Kali: `sudo apt-get install libgeoip-dev`

OpenSUSE: `sudo zypper install libGeoIP-devel`

Arch: `sudo pacman -S geoip`
libmaxminddb can be found in the Arch User Repository (AUR) at
https://aur.archlinux.org/packages/libmaxminddb.

Mac OS X: `brew install geoip libmaxminddb`

16.2.1 Databases Update

The geoIP databases can be updated with the `updatedb.sh` script as follows:

```
./scripts/updatedb.sh
```

Alternatively the latest version of the databases can be found at https://dev.maxmind.com/geoip/geoip2/geolite2/ (GeoLite2-City). Legacy databases, the latest version of which can be found at https://dev.maxmind.com/geoip/legacy/geolite (Geo Lite City and Geo Lite City IPv6), are also supported.

16.3 Configuration Flags

The following flags can be used to control the output of the plugin (Information in italic only applies to legacy databases):

Name	Default	Description
GEOIP_LEGACY	0	Whether to use GeoLite2 (0) or the GeoLite legacy database (1)
GEOIP_SRC	1	Display geo info for the source IP
GEOIP_DST	1	Display geo info for the destination IP
GEOIP_CONTINENT	2	0: no continent, 1: name (**GeoLite2**), 2: two letters code
GEOIP_COUNTRY	2	0: no country, 1: name, 2: two letters code, *3: three letters code*

Name	Default	Description
GEOIP_REGION	*1*	*0: no region, 1: name, 2: code*
GEOIP_CITY	1	Display the city of the IP
GEOIP_POSTCODE	1	Display the postal code of the IP
GEOIP_ACCURACY	1	(**GeoLite2**) Display the accuracy of the geolocation
GEOIP_POSITION	1	Display the position (latitude, longitude) of the IP
GEOIP_METRO_CODE	0	Display the metro (dma) code of the IP (US only)
GEOIP_AREA_CODE	*0*	*Display the telephone area code of the IP*
GEOIP_NETMASK	*1*	*0: no netmask, 1: netmask as int (cidr), 2: netmask as hex, 3: netmask as IP*
GEOIP_TIME_ZONE	1	(**GeoLite2**) Display the time zone
GEOIP_LANG	"en"	(**GeoLite2**) Language to use: Brazilian Portuguese (pt-BR), English (en), French (fr), German (de), Japanese (jp), Russian (ru), Simplified Chinese (zh-CN) or Spanish (es)
GEOIP_BUFSIZE	64	(**GeoLite2**) Buffer size
GEOIP_DB_CACHE	*2*	*0: read DB from file system (slower, least memory)* *1: index cache (cache frequently used index only)* *2: memory cache (faster, more memory)*
GEOIP_UNKNOWN	"--"	Representation of unknown locations (GeoIP's default)

16.4 Flow File Output

The geoip plugin outputs the following columns (for src and dst IP):

Column	Type	Description	Flags
srcIpContinent	S	Continent name	GEOIP_CONTINENT=1
srcIpContinent	SC	Continent code	GEOIP_CONTINENT=2
srcIpCountry	S	Country name	GEOIP_COUNTRY=1
srcIpCountry	SC	Country code	GEOIP_COUNTRY=2|3
srcIpRegion	SC	Region	GEOIP_REGION=1
srcIpRegion	S	Region	GEOIP_REGION=2
srcIpCity	S	City	
srcIpPostcode	SC	Postal code	
srcIpAccuracy	U16	Accuracy of the geolocation (in km)	
srcIpLatitude	D	Latitude	GEOIP_LEGACY=0
srcIpLongitude	D	Longitude	GEOIP_LEGACY=0
srcIpLatitude	F	Latitude	GEOIP_LEGACY=1
srcIpLongitude	F	Longitude	GEOIP_LEGACY=1
srcIpMetroCode	U16	Metro (DMA) code (US only)	GEOIP_LEGACY=0
srcIpMetroCode	I32	Metro (DMA) code (US only)	GEOIP_LEGACY=1
srcIpAreaCode	I32	Area code	
srcIpNetmask	U32	Netmask (CIDR)	GEOIP_NETMASK=1
srcIpNetmask	H32	Netmask	GEOIP_NETMASK=2

Column	Type	Description	Flags
srcIpNetmask	IP4	Netmask	GEOIP_NETMASK=3
srcIpTimeZone	S	Time zone	
geoStat	H8	Status	GEOIP_LEGACY=0

16.4.1 srcIpContinent

Continent codes are as follows:

Code	Description
AF	Africa
AS	Asia
EU	Europe
NA	North America
OC	Oceania
SA	South America
--	Unknown (see GEOIP_UNKNOWN)

16.4.2 geoStat

The geoStat column is to be interpreted as follows:

geoStat	Description
2^0 (=0x01)	A string had to be truncated... increase GEOIP_BUFSIZE

16.5 Post-Processing

The geoIP plugin comes with the genkml.sh script which generates a KML (Keyhole Markup Language) file from a flow file. This KML file can then be loaded in Google Earth to display the location of the IP addresses involved in the dump file. Its usage is straightforward:

```
./scripts/genkml.sh FILE_flows.txt
```

17 httpSniffer

The httpSniffer plugin processes HTTP header and content information of a flow. The idea is to identify certain HTTP features using flow parameters and to extract certain content such as text or images for further investigation. The httpSniffer plugin requires no dependencies and produces only output to the flow file. User defined compiler switches in *httpSniffer.h* produce optimized code for the specific application.

17.1 Configuration Flags

The flow based output and the extracted information can be controlled by switches and constants listed in the table below. They control the output of host, URL and method counts, names and cookies and the function of content storage. **WARNING:** The amount of being stored on disk can be substantial, make sure that the number of concurrent file handles is large enough, use `ulimit -n`.

Name	Default	Description	Flags
HTTP_MIME	1	mime types	
HTTP_STAT	1	status codes	
HTTP_MCNT	1	mime count: get, post	
HTTP_HOST	1	hosts	
HTTP_URL	1	URLs	
HTTP_COOKIE	1	cookies	
HTTP_IMAGE	1	image names	
HTTP_VIDEO	1	video names	
HTTP_AUDIO	1	audio names	
HTTP_MSG	1	message names	
HTTP_APPL	1	application names	
HTTP_TEXT	1	text names	
HTTP_PUNK	1	post/else/unknown names	
HTTP_BODY	1	analyse body and print anomalies	
HTTP_BDURL	1	refresh and set-cookie URLs	HTTP_BODY=1
HTTP_USRAG	1	user agents	
HTTP_XFRWD	1	X-Forward	
HTTP_REFRR	1	Referer	
HTTP_VIA	1	Via	
HTTP_LOC	0	Location	
HTTP_SERV	1	Server	
HTTP_PWR	1	Powered by	
HTTP_STATA	1	aggregate status response codes	
HTTP_HOSTAGA	1	aggregate hosts	
HTTP_URLAGA	1	aggregate URLs	
HTTP_USRAGA	1	aggregate user agents	
HTTP_XFRWDA	1	aggregate X-Forward-For	
HTTP_REFRRA	1	aggregate Referer	
HTTP_VIAA	1	aggregate Via	
HTTP_LOCA	1	aggregate Location	
HTTP_SERVA	1	aggregate Server	
HTTP_PWRA	1	aggregate Powered by	

Name	Default	Description	Flags
HTTP_SAVE_IMAGE	0	save all images	
HTTP_SAVE_VIDEO	0	save all videos	
HTTP_SAVE_AUDIO	0	save all audios	
HTTP_SAVE_MSG	0	save all messages	
HTTP_SAVE_TEXT	0	save all texts	
HTTP_SAVE_APPL	0	save all applications	
HTTP_SAVE_PUNK	0	save all else	
HTTP_RM_PICDIR	0	delete directories at T2 start	

Aggregate mode is on by default to save memory space. Note that HTTP_SAVE_* refers to the *Content-Type*, e.g., HTTP_SAVE_APPL, will save all payload whose Content-Type starts with application/ (including forms, such as application/x-www-form-urlencoded). The maximum memory allocation per item is defined by HTTP_DATA_C_MAX listed below. The path of each extracted http content can be set by the HTTP_XXXX_PATH constant. HTTP content having no name is assigned a default name defined by HTTP_NONAME_IMAGE. Each name is appended by the findex, packet number and an index to facilitate the mapping between flows and its content. The latter constant has to be chosen carefully because for each item: mime, cookie, image, etc, HTTP_MXFILE_LEN * HTTP_DATA_C_MAX * HASHCHAINTABLE_SIZE * HASHFACTOR bytes are allocated. The filenames are defined as follows:
Filename_Flow-Dir(0/1)_findex_#Packet-in-Flow_#Mimetype-in-Flow
So they can easily being matched with the flow or packet file. If the flow containing the filename is not present Filename = HTTP_NONAME, defined in httpSniffer.h.

Name	Default	Description
HTTP_PATH	"/tmp/"	Root path
HTTP_IMAGE_PATH	HTTP_PATH"httpPicture/"	Path for pictures
HTTP_VIDEO_PATH	HTTP_PATH"httpVideo/"	Path for videos
HTTP_AUDIO_PATH	HTTP_PATH"httpAudio/"	Path for audios
HTTP_MSG_PATH	HTTP_PATH"httpMSG/"	Path for messages
HTTP_TEXT_PATH	HTTP_PATH"httpText/"	Path for texts
HTTP_APPL_PATH	HTTP_PATH"httpAppl/"	Path for applications
HTTP_PUNK_PATH	HTTP_PATH"httpPunk/"	Path for put/else
HTTP_NONAME_IMAGE	"nudel"	File name for unnamed content
HTTP_DATA_C_MAX	20	Maximum dim of all storage array: # / flow
HTTP_CNT_LEN	13	max # of cnt digits attached to file name
HTTP_FINDEX_LEN	20	string length of findex in decimal format.
HTTP_MXFILE_LEN	80	Maximum image name length in bytes
HTTP_MXUA_LEN	400	Maximum user agent name length in bytes
HTTP_MXXF_LEN	80	Maximum x-forward-for name length in bytes

17.2 Flow File Output

The default settings will result in six tab separated columns in the flow file where the items in column 4-6 are sequences of strings separated by ';'. Whereas an item switch is set to '0' only the occurrence of this item during the flow is supplied. It is a high speed mode for large datasets or real-time operation in order to produce an initial idea of interesting flows

maybe by script based post processing selecting also by the information supplied by first three columns.

Column	Type	Description	Flags
httpStat	H16	Status	
httpAFlags	H16	Anomaly flags	
httpMethods	H8	HTTP methods	
httpHeadMimes	H16	HEADMIME-TYPES	
httpCFlags	H8	HTTP content body info	HTTP_BODY=1
httpGet_Post	2U16	Number of GET and POST requests	HTTP_MCNT=1
httpRSCnt	U16	Response status count	HTTP_STAT=1
httpRSCode	RU16	Response status code	HTTP_STAT=1
httpURL_Via_Loc_Srv_ Pwr_UAg_XFr_ Ref_Cky_Mim	10U16	Number of URL, Via, Location, Server, Powered-By, User-Agent, X-Forwarded-For, Referer, Cookie and Mime-Type	
httpImg_Vid_Aud_Msg_ Txt_App_Unk	7U16	Number of images, videos, audios, messages, texts, applications and unknown	
httpHosts	RS	Host names	HTTP_HOST=1
httpURL	RS	URLs (including parameters)	HTTP_URL=1
httpMimes	RS	MIME-types	HTTP_MIME=1
httpCookies	RS	Cookies	HTTP_COOKIE=1
httpImages	RS	Images	HTTP_IMAGE=1
httpVideos	RS	Videos	HTTP_VIDEO=1
httpAudios	RS	Audios	HTTP_AUDIO=1
httpMsgs	RS	Messages	HTTP_MSG=1
httpAppl	RS	Applications	HTTP_APPL=1
httpText	RS	Texts	HTTP_TEXT=1
httpPunk	RS	Punk	HTTP_PUNK=1
httpBdyURL	RS	Body: Refresh, set_cookie URL	HTTP_BODY=1&& HTTP_BDURL=1
httpUsrAg	RS	User-Agent	HTTP_USRAG=1
httpXFor	RS	X-Forwarded-For	HTTP_XFRWD=1
httpRefrr	RS	Referer	HTTP_REFRR=1
httpVia	RS	Via (Proxy)	HTTP_VIA=1
httpLoc	RS	Location (Redirection)	HTTP_LOC=1
httpServ	RS	Server	HTTP_SERV=1
httpPwr	RS	Powered-By / Application	HTTP_PWR=1

17.2.1 httpStat

The httpStat column is to be interpreted as follows:

httpStat	Description
2^0 (=0x0001)	Warning: HTTP_DATA_C_MAX entries in flow name array reached
2^1 (=0x0002)	Warning: Filename longer than HTTP_MXFILE_LEN
2^2 (=0x0004)	Internal State: pending url name
2^3 (=0x0008)	HTTP Flow

httpStat	Description
2^4 (=0x0010)	Internal State: Chunked transfer
2^5 (=0x0020)	Internal State: HTTP Flow detected
2^6 (=0x0040)	Internal State: http header parsing in process
2^7 (=0x0080)	Internal State: sequence number init
2^8 (=0x0100)	Internal State: header shift
2^9 (=0x0200)	Internal State: PUT payload sniffing
2^{10} (=0x0400)	Internal State: Image payload sniffing
2^{11} (=0x0800)	Internal State: video payload sniffing
2^{12} (=0x1000)	Internal State: audio payload sniffing
2^{13} (=0x2000)	Internal State: message payload sniffing
2^{14} (=0x4000)	Internal State: text payload sniffing
2^{15} (=0x8000)	Internal State: application payload sniffing

17.2.2 httpAFlags

The `httpAFlags` column denotes HTTP anomalies regarding the protocol and the security. It is to be interpreted as follows:

httpAFlags	Description
2^0 (=0x0001)	Warning: POST query with parameters, possible malware
2^1 (=0x0002)	Warning: Host is IPv4
2^2 (=0x0004)	Warning: Possible DGA
2^3 (=0x0008)	Warning: Mismatched content-type
2^4 (=0x0010)	Warning: Sequence number mangled or error retry detected
2^5 (=0x0020)	Warning: Parse Error
2^6 (=0x0040)	Warning: header without value, e.g., `Content-Type: [missing]`
2^7 (=0x0080)	
2^8 (=0x0100)	Info: X-Site Scripting protection
2^9 (=0x0200)	Info: Content Security Policy
2^{10} (=0x0400)	
2^{11} (=0x0800)	
2^{12} (=0x1000)	Warning: possible exe download, check also mime type for conflict
2^{13} (=0x2000)	Warning: possible ELF download, check also mime type for conflict
2^{14} (=0x4000)	Warning: HTTP 1.0 legacy protocol, often used by malware
2^{15} (=0x8000)	

17.2.3 httpMethods

The aggregated `httpMethods` bit field provides an instant overview about the protocol state and communication during a flow. It can also be used during post processing in order to select only flows containing e.g. responses or delete operations.

httpMethods	Type	Description
(=0x00)	RESPONSE	Response of server identified by URL
2^0 (=0x01)	OPTIONS	Return HTTP methods that server supports for specified URL

httpMethods	Type	Description
2^1 (=0x02)	GET	Request of representation of specified resource
2^2 (=0x04)	HEAD	Request of representation of specified resource without BODY
2^3 (=0x08)	POST	Request to accept enclosed entity as new subordinate of resource identified by URI
2^4 (=0x10)	PUT	Request to store enclosed entity under supplied URI
2^5 (=0x20)	DELETE	Delete specified resource
2^6 (=0x40)	TRACE	Echo back received request
2^7 (=0x80)	CONNECT	Convert request connection to transparent TCP/IP tunnel

17.2.4 httpHeadMimes

The aggregated `httpHeadMimes` bit field provides an instant overview about the content of the HTTP payload being transferred during a flow. Thus, the selection of flows with certain content during post processing is possible even when the plugin is set to count mode for all items in order to conserve memory and processing capabilities. The 16 Bit information is separated into Mime Type (MT) and Common Subtype Prefixes (CSP) / special Flags each comprising of 8 Bit. This is experimental and is subject to change if a better arrangement is found.

httpHeadMimes	MT / CSP	Description
2^0 (=0x0001)	application	Multi-purpose files: java or post script, etc
2^1 (=0x0002)	audio	Audio file
2^2 (=0x0004)	image	Image file
2^3 (=0x0008)	message	Instant or email message type
2^4 (=0x0010)	model	3D computer graphics
2^4 (=0x0020)	multipart	Archives and other objects made of more than one part
2^5 (=0x0040)	text	Human-readable text and source code
2^6 (=0x0080)	video	Video stream: Mpeg, Flash, Quicktime, etc
2^8 (=0x0100)	vnd	vendor-specific files: Word, OpenOffice, etc
2^9 (=0x0200)	x	Non-standard files: tar, SW packages, LaTex, Shockwave Flash, etc
2^{10} (=0x0400)	x-pkcs	public-key cryptography standard files
2^{11} (=0x0800)	—	—
2^{12} (=0x1000)	pdf	—
2^{13} (=0x2000)	java	—
2^{14} (=0x4000)	—	—
2^{15} (=0x8000)	allelse	All else

17.2.5 httpCFlags

The `httpCFlags` contain information about the content body, regarding to information about rerouting. They have to be interpreted as follows:

httpBodyFlags	MT / CSP	Description
2^0 (=0x0001)	STCOOKIE	http set cookie
2^1 (=0x0002)	REFRESH	http refresh detected
2^2 (=0x0004)	HOSTNAME	host name detected
2^3 (=0x0008)	BOUND	Post Boundary marker

httpBodyFlags	MT / CSP	Description
2^4 (=0x0010)	PCNT	Potential HTTP content
2^5 (=0x0020)	—	
2^6 (=0x0040)	QUARA	Quarantine Virus upload
2^{15} (=0x8000)	—	

17.3 Plugin Report Output

The following information is reported:

- Max number of file handles (only if `HTTP_SAVE=1`)
- Number of HTTP IPv4/6 packets
- Number of HTTP #GET, #POST, #GET/#POST ratio
- Aggregated status flags (`httpStat`)
- Aggregated mimetype flags (`httpHeadMimes`)
- Aggregated anomaly flags (`httpAFlags`)
- Aggregated content flags (`httpCFlags`, only if `HTTP_BODY=1`)

The GET/POST ratio is very helpful in detecting malware operations, if you know the normal ratio of your machines in the network. The file descriptor gives you an indication of the maximum file handles the present pcap will produce. You can increase it by invoking `uname -n mylimit`, but it should not be necessary as we manage the number of handle being open to be always below the max limit.

18 icmpDecode

18.1 Description

The icmpDecode plugin analyzes ICMP and ICMPv6 traffic. It generates global and flow based statistics.

18.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description	Flags
ICMP_TC_MD	0	0: Type/code as bitfield 1: Type/code as explicit array	
ICMP_NUM	10	Number of type and code information	ICMP_TC_MD=1
ICMP_FDCORR	1	Flow direction correction	
ICMP_PARENT	0	Whether (1) or not (0) to resolve the parent flow	
ICMP_STATFILE	0	Whether (1) or not (0) to print ICMP statistics in a file	

18.3 Flow File Output

The icmpDecode plugin outputs the following columns:

Column	Type	Description	Flags
icmpStat	H8	Status	
icmpTCcnt	U8	type code count	
icmpBFType_Code	H32_H16	Aggregated type (<32) and code bitfield	ICMP_TC_MD=0&& IPV6_ACTIVATE=0
icmpBFTypH_TypL_Code	H32_H32_H16	Aggr. type (H>128), L(<32) and code bitfield	ICMP_TC_MD=0&& IPV6_ACTIVATE=1
icmpType_Code	R(U8_U8)	Type and code fields	ICMP_TC_MD=1
icmpTmGtw	H32	Time/gateway	
icmpEchoSuccRatio	F	Echo reply/request success ratio	
icmpPFindex	U64	Parent flowIndex	ICMP_PARENT=1

18.3.1 icmpStat

The icmpStat column is to be interpreted as follows:

icmpStat	Description
2^0 (=0x01)	Flow is ICMP
2^1 (=0x02)	—
2^2 (=0x04)	—
2^3 (=0x08)	—
2^4 (=0x10)	WANG2 Microsoft bandwidth test
2^5 (=0x20)	—
2^6 (=0x40)	—

icmpStat	Description
2^7 (=0x80)	—

18.3.2 icmpBFType_Code

For **ICMP (IPv4)**, the icmpBFType_Code column is to be interpreted as follows:

icmpBFType	Description
2^0 (=0x00000001)	Echo Reply
2^1 (=0x00000002)	—
2^2 (=0x00000004)	—
2^3 (=0x00000008)	Destination Unreachable
2^4 (=0x00000010)	Source Quench
2^5 (=0x00000020)	Redirect (change route)
2^6 (=0x00000040)	—
2^7 (=0x00000080)	Echo Request
2^8 (=0x00000100)	—
2^9 (=0x00000200)	—
2^{10} (=0x00000400)	—
2^{11} (=0x00000800)	Time Exceeded
2^{12} (=0x00001000)	Parameter Problem
2^{13} (=0x00002000)	Timestamp Request
2^{14} (=0x00004000)	Timestamp Reply
2^{15} (=0x00008000)	Information Request

icmpBFType	Description
2^{16} (=0x00010000)	Information Reply
2^{17} (=0x00020000)	Address Mask Request
2^{18} (=0x00040000)	Address Mask Reply
2^{19} (=0x00080000)	—
2^{20} (=0x00100000)	—
2^{21} (=0x00200000)	—
2^{22} (=0x00400000)	—
2^{23} (=0x00800000)	—
2^{24} (=0x01000000)	—
2^{25} (=0x02000000)	—
2^{26} (=0x04000000)	—
2^{27} (=0x08000000)	—
2^{28} (=0x10000000)	—
2^{29} (=0x20000000)	—
2^{30} (=0x40000000)	Traceroute
2^{31} (=0x80000000)	—

The icmpCode for **Destination Unreachable** (0x00000008) is to be interpreted as follows:

icmpBFCode	Description
2^0 (=0x0001)	Network Unreachable
2^1 (=0x0002)	Host Unreachable
2^2 (=0x0004)	Protocol Unreachable
2^3 (=0x0008)	Port Unreachable
2^4 (=0x0010)	Fragmentation Needed/DF set
2^5 (=0x0020)	Source Route failed
2^6 (=0x0040)	—
2^7 (=0x0080)	—

icmpBFCode	Description
2^8 (=0x0100)	—
2^9 (=0x0200)	—
2^{10} (=0x0400)	—
2^{11} (=0x0800)	—
2^{12} (=0x1000)	—
2^{13} (=0x2000)	Packet filtered
2^{14} (=0x4000)	Precedence violation
2^{15} (=0x8000)	Precedence cut off

For **ICMPv6 (IPv6)**, the `icmpBFType_Code` column is to be interpreted as follows:

icmpType	Description
0	Reserved
1	Destination Unreachable
2	Packet Too Big
3	Time Exceeded
4	Parameter Problem
100	Private experimentation
101	Private experimentation
102–126	Unassigned
127	Reserved for expansion of ICMPv6 error messages
128	Echo Request
129	Echo Reply
130	Multicast Listener Query
131	Multicast Listener Report
132	Multicast Listener Done
133	Router Solicitation
134	Router Advertisement
135	Neighbor Solicitation
136	Neighbor Advertisement
137	Redirect Message
138	Router Renumbering
139	ICMP Node Information Query
140	ICMP Node Information Response
141	Inverse Neighbor Discovery Solicitation
142	Inverse Neighbor Discovery Advertisement
143	Version 2 Multicast Listener Report
144	Home Agent Address Discovery Request
145	Home Agent Address Discovery Reply
146	Mobile Prefix Solicitation
147	Mobile Prefix Advertisement
148	Certification Path Solicitation
149	Certification Path Advertisement
150	ICMP messages utilized by experimental mobility protocols such as Seamoby
151	Multicast Router Advertisement
152	Multicast Router Solicitation
153	Multicast Router Termination
154	FMIPv6 Messages
155	RPL Control Message
156	ILNPv6 Locator Update Message
157	Duplicate Address Request
158	Duplicate Address Confirmation
159	MPL Control Message
160–199	Unassigned
200	Private experimentation
201	Private experimentation
255	Reserved for expansion of ICMPv6 informational messages

The `icmpCode` for **Destination Unreachable (1)** are:

icmpCode	Description
2^0 (=0x0001)	No route to destination
2^1 (=0x0002)	Communication with destination administratively prohibited
2^2 (=0x0004)	Beyond scope of source address
2^3 (=0x0008)	Address unreachable
2^4 (=0x0010)	Port unreachable
2^5 (=0x0020)	Source address failed ingress/egress policy
2^6 (=0x0040)	Reject route to destination
2^7 (=0x0080)	Error in Source Routing Header

The `icmpCode` for **Time Exceeded (3)** are:

icmpCode	Description
2^0 (=0x0001)	Hop limit exceeded in transit
2^1 (=0x0002)	Fragment reassembly time exceeded

The `icmpCode` for **Parameter Problem (4)** are:

icmpCode	Description
2^0 (=0x0001)	Erroneous header field encountered
2^1 (=0x0002)	Unrecognized Next Header type encountered
2^2 (=0x0004)	Unrecognized IPv6 option encountered
2^3 (=0x0008)	IPv6 First Fragment has incomplete IPv6 Header Chain

The `icmpCode` for **Router Renumbering (138)** are:

icmpCode	Description
2^0 (=0x0001)	Router Renumbering Command
2^1 (=0x0002)	Router Renumbering Result
255	Sequence Number Reset

The `icmpCode` for **ICMP Node Information Query (139)** are:

icmpCode	Description
2^0 (=0x0001)	The Data field contains an IPv6 address which is the Subject of this Query
2^1 (=0x0002)	The Data field contains a name which is the Subject of this Query, or is empty, as in the case of a NOOP
2^3 (=0x0004)	The Data field contains an IPv4 address which is the Subject of this Query

The `icmpCode` for **ICMP Node Information Response (140)** are:

icmpCode	Description
2^0 (=0x0001)	A successful reply. The Reply Data field may or may not be empty
2^1 (=0x0002)	The Responder refuses to supply the answer. The Reply Data field will be empty
2^2 (=0x0004)	The Qtype of the Query is unknown to the Responder. The Reply Data field will be empty

18.4 Packet File Output

In packet mode (`-s` option), the icmpDecode plugin outputs the following columns:

Column	Type	Description	Flags
icmpType	U8	Message type	
icmpCode	U8	Message code	
icmpPFindex	U64	Parent flowIndex	ICMP_PARENT=1

18.5 Additional Output

The icmpDecode plugin outputs absolute and relative statistics in the `PREFIX_icmpStats.txt` file. Note that the default suffix of "`_icmpStats.txt`" can be changed by editing the `ICMP_SUFFIX` flag.

The output is as follows (`IPV6_ACTIVATE=0||IPV6_ACTIVATE=2`):

Type	Code	Description
ICMP_ECHOREQUEST	—	Echo request
ICMP_ECHOREPLY	—	Echo reply to an echo request
ICMP_SOURCE_QUENCH	—	Source quenches
ICMP_TRACEROUTE	—	Traceroute packets
ICMP_DEST_UNREACH	ICMP_NET_UNREACH	Network unreachable
ICMP_DEST_UNREACH	ICMP_HOST_UNREACH	Host unreachable
ICMP_DEST_UNREACH	ICMP_PROT_UNREACH	Protocol unreachable
ICMP_DEST_UNREACH	ICMP_PORT_UNREACH	Port unreachable
ICMP_DEST_UNREACH	ICMP_FRAG_NEEDED	Fragmentation needed
ICMP_DEST_UNREACH	ICMP_SR_FAILED	Source route failed
ICMP_DEST_UNREACH	ICMP_NET_UNKNOWN	Network unknown
ICMP_DEST_UNREACH	ICMP_HOST_UNKNOWN	Host unknown
ICMP_DEST_UNREACH	ICMP_HOST_ISOLATED	Host is isolated
ICMP_DEST_UNREACH	ICMP_NET_ANO	Network annotation
ICMP_DEST_UNREACH	ICMP_HOST_ANO	Host annotation
ICMP_DEST_UNREACH	ICMP_NET_UNR_TOS	Unreachable type of network service
ICMP_DEST_UNREACH	ICMP_HOST_UNR_TOS	Unreachable type of host service
ICMP_DEST_UNREACH	ICMP_PKT_FILTERED	Dropped by a filtering device
ICMP_DEST_UNREACH	ICMP_PREC_VIOLATION	Precedence violation
ICMP_DEST_UNREACH	ICMP_PREC_CUTOFF	Precedence cut off
ICMP_REDIRECT	ICMP_REDIR_NET	Network redirection
ICMP_REDIRECT	ICMP_REDIR_HOST	Host redirection
ICMP_REDIRECT	ICMP_REDIR_NETTOS	Network type of service
ICMP_REDIRECT	ICMP_REDIR_HOSTTOS	Host type of service
ICMP_TIME_EXCEEDED	ICMP_EXC_TTL	TTL exceeded in Transit
ICMP_TIME_EXCEEDED	ICMP_EXC_FRAGTIME	Fragment Reassembly Time Exceeded

If `IPV6_ACTIVATE>0`, then the output becomes:

Type	Code	Description
ICMP6_ECHOREQUEST	—	Echo request
ICMP6_ECHOREPLY	—	Echo reply to an echo request
ICMP6_PKT_TOO_BIG	—	Packet too big
ICMP6_DEST_UNREACH	ICMP6_NO_ROUTE	No route to destination
ICMP6_DEST_UNREACH	ICMP6_COMM_PROHIBIT	Communication with destination prohibited
ICMP6_DEST_UNREACH	ICMP6_BEYOND_SCOPE	Beyond scope of source address
ICMP6_DEST_UNREACH	ICMP6_ADDR_UNREACH	Address unreachable
ICMP6_DEST_UNREACH	ICMP6_PORT_UNREACH	Port unreachable
ICMP6_DEST_UNREACH	ICMP6_SR_FAILED	Source route failed

Type	Code	Description
ICMP6_DEST_UNREACH	ICMP6_REJECT	Reject source to destination
ICMP6_DEST_UNREACH	ICMP6_ERROR_HDR	Error in Source Routing Header
ICMP6_TIME_EXCEEDED	ICMP6_EXC_HOPS	Hop limit exceeded in transit
ICMP6_TIME_EXCEEDED	ICMP6_EXC_FRAGTIME	Fragment reassembly time exceeded
ICMP6_PARAM_PROBLEM	ICMP6_ERR_HDR	Erroneous header field
ICMP6_PARAM_PROBLEM	ICMP6_UNRECO_NEXT_HDR	Unrecognized Next Header type
ICMP6_PARAM_PROBLEM	ICMP6_UNRECO_IP6_OPT	Unrecognized IPv6 option
ICMP6_MCAST_QUERY	—	Multicast Listener Query
ICMP6_MCAST_REP	—	Multicast Listener Report
ICMP6_MCAST_DONE	—	Multicast Listener Done
ICMP6_RTER_SOLICIT	—	Router Solicitation
ICMP6_RTER_ADVERT	—	Router Advertisement
ICMP6_NBOR_SOLICIT	—	Neighbor Solicitation
ICMP6_NBOR_ADVERT	—	Neighbor Advertisement
ICMP6_REDIRECT_MSG	—	Redirect Message
ICMP6_RTER_RENUM	ICMP6_RR_CMD (0)	Router Renumbering Command
ICMP6_RTER_RENUM	ICMP6_RR_RES (1)	Router Renumbering Result
ICMP6_RTER_RENUM	ICMP6_RR_RST (255)	Router Renum.: Sequence Number Reset
ICMP6_NODE_INFO_QUERY	ICMP6_NIQ_IP6 (0)	Node Info. Query: contains an IPv6 address
ICMP6_NODE_INFO_QUERY	ICMP6_NIQ_NAME (1)	Contains a name or is empty (NOOP)
ICMP6_NODE_INFO_QUERY	ICMP6_NIQ_IP4 (2)	Contains an IPv4 address
ICMP6_NODE_INFO_RESP	ICMP6_NIR_SUCC (0)	Node Info. Response: Successful reply
ICMP6_NODE_INFO_RESP	ICMP6_NIR_DENIED (1)	Responder refuses to answer
ICMP6_NODE_INFO_RESP	ICMP6_NIR_UNKN (2)	Qtype of the query unknown
ICMP6_INV_NBOR_DSM	—	Inverse Neighbor Discovery Solicitation Msg
ICMP6_INV_NBOR_DAM	—	Inverse Neighbor Disc. Advertisement Msg
ICMP6_MLD2	—	Version 2 Multicast Listener Report
ICMP6_ADDR_DISC_REQ	—	Home Agent Address Discovery Request Msg
ICMP6_ADDR_DISC_REP	—	Home Agent Address Discovery Reply Msg
ICMP6_MOB_PREF_SOL	—	Mobile Prefix Solicitation
ICMP6_MOB_PREF_ADV	—	Mobile Prefix Advertisement
ICMP6_CERT_PATH_SOL	—	Certification Path Solicitation Message
ICMP6_CERT_PATH_ADV	—	Certification Path Advertisement Message
ICMP6_EXP_MOBI	—	Experimental mobility protocols
ICMP6_MRD_ADV	—	Multicast Router Advertisement
ICMP6_MRD_SOL	—	Multicast Router Solicitation
ICMP6_MRD_TERM	—	Multicast Router Termination
ICMP6_FMIPV6	—	FMIPv6 Messages
ICMP6_RPL_CTRL	—	RPL Control Message
ICMP6_ILNP_LOC_UP	—	ILNPv6 Locator Update Message
ICMP6_DUP_ADDR_REQ	—	Duplicate Address Request
ICMP6_DUP_ADDR_CONF	—	Duplicate Address Confirmation

18.6 Post-Processing

18.6.1 icmpX

The `icmpX` script extracts all ICMP flows and their parents (flows which caused the ICMP message) from a flow file. Run `./icmpX --help` for more information.

19 igmpDecode

This plugin analyzes IGMP traffic and provides absolute and relative statistics to the `PREFIX_igmpStats.txt` file.

19.1 Required Files

None

19.2 Flow File Output

The igmpDecode plugin outputs the following columns:

Column	Type	Description
igmpStat	H8	Status
igmpVersion	RI8	Version
igmpAType	H32	Aggregated type
igmpMCastAddr	IP4	Multicast address
igmpNRec	U16	# of records

19.2.1 igmpStat

The `igmpStat` column is to be interpreted as follows:

igmpStat	Description
2^0 (=0x01)	IGMP message had invalid length
2^1 (=0x02)	IGMP message had invalid checksum
2^2 (=0x04)	IGMP message had invalid TTL ($\neq 1$)
2^3 (=0x08)	IGMP message was invalid for other reasons

19.3 Additional Output

The plugin exports global statistics about IGMP traffic in the `PREFIX_igmpStats.txt` file.

20 ircDecode

20.1 Description

The ircDecode plugin analyses IRC traffic. User defined compiler switches are in *ircDecode.h*.

20.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
IRC_SAVE	0	Save content to IRC_F_PATH
IRC_BITFIELD	0	Bitfield coding of IRC commands
IRC_UXNMLN	10	maximal USER length
IRC_PXNMLN	10	maximal PW length
IRC_MXNMLN	50	maximal name length
IRC_MAXUNM	5	Maximal number of users
IRC_MAXPNM	5	Maximal number of passwords
IRC_MAXCNM	20	Maximal number of parameters

20.3 Flow File Output

The ircDecode plugin outputs the following columns:

Column	Type	Description	Flags
ircStat	H8	Status	
ircCBF	H64	Commands	BITFIELD=1
ircCC	RSC	Command codes	
ircRC	RU16	Response codes	
ircUsrNum	U8	Number of users	
ircPwNum	U8	Number of passwords	
ircCNum	U8	Number of parameters	
ircUsr	RS	Users	
ircPw	RS	Passwords	
ircC	RS	Content	

20.3.1 ircStat

The ircStat column is to be interpreted as follows:

ircStat	Description
2^0 (=0x01)	IRC port found
2^1 (=0x02)	IRC passive parent flow
2^2 (=0x04)	IRC passive write finished
2^3 (=0x08)	IRC active parent flow
2^4 (=0x10)	—

ircStat	Description
2^5 (=0x20)	File error
2^6 (=0x40)	—
2^7 (=0x80)	Array overflow

20.3.2 ircCBF

The `ircCBF` column is to be interpreted as follows:

ircCBF	Description
2^0 (=0x0000.0000.0000.0001)	ADMIN
2^1 (=0x0000.0000.0000.0002)	AWAY
2^2 (=0x0000.0000.0000.0004)	CONNECT
2^3 (=0x0000.0000.0000.0008)	DIE
2^4 (=0x0000.0000.0000.0010)	ERROR
2^5 (=0x0000.0000.0000.0020)	INFO
2^6 (=0x0000.0000.0000.0040)	INVITE
2^7 (=0x0000.0000.0000.0080)	ISON
2^8 (=0x0000.0000.0000.0100)	JOIN
2^9 (=0x0000.0000.0000.0200)	KICK
2^{10} (=0x0000.0000.0000.0400)	KILL
2^{11} (=0x0000.0000.0000.0800)	LINKS
2^{12} (=0x0000.0000.0000.1000)	LIST
2^{13} (=0x0000.0000.0000.2000)	LUSERS
2^{14} (=0x0000.0000.0000.4000)	MODE
2^{15} (=0x0000.0000.0000.8000)	MOTD
2^{16} (=0x0000.0000.0001.0000)	NAMES
2^{17} (=0x0000.0000.0002.0000)	NICK
2^{18} (=0x0000.0000.0004.0000)	NJOIN
2^{19} (=0x0000.0000.0008.0000)	NOTICE
2^{20} (=0x0000.0000.0010.0000)	OPER
2^{21} (=0x0000.0000.0020.0000)	PART
2^{22} (=0x0000.0000.0040.0000)	PASS
2^{23} (=0x0000.0000.0080.0000)	PING
2^{24} (=0x0000.0000.0100.0000)	PONG
2^{25} (=0x0000.0000.0200.0000)	PRIVMSG
2^{26} (=0x0000.0000.0400.0000)	QUIT
2^{27} (=0x0000.0000.0800.0000)	REHASH
2^{28} (=0x0000.0000.1000.0000)	RESTART
2^{29} (=0x0000.0000.2000.0000)	SERVER
2^{30} (=0x0000.0000.4000.0000)	SERVICE

ircCBF	Description
2^{31} (=0x0000.0000.8000.0000)	SERVLIST
2^{32} (=0x0000.0001.0000.0000)	SQUERY
2^{33} (=0x0000.0002.0000.0000)	SQUIRT
2^{34} (=0x0000.0004.0000.0000)	SQUIT
2^{35} (=0x0000.0008.0000.0000)	STATS
2^{36} (=0x0000.0010.0000.0000)	SUMMON
2^{37} (=0x0000.0020.0000.0000)	TIME
2^{38} (=0x0000.0040.0000.0000)	TOPIC
2^{39} (=0x0000.0080.0000.0000)	TRACE
2^{40} (=0x0000.0100.0000.0000)	USER
2^{41} (=0x0000.0200.0000.0000)	USERHOST
2^{42} (=0x0000.0400.0000.0000)	USERS
2^{43} (=0x0000.0800.0000.0000)	VERSION
2^{44} (=0x0000.1000.0000.0000)	WALLOPS
2^{45} (=0x0000.2000.0000.0000)	WHO
2^{46} (=0x0000.4000.0000.0000)	WHOIS
2^{47} (=0x0000.8000.0000.0000)	WHOWAS
2^{48} (=0x0001.0000.0000.0000)	-
2^{49} (=0x0002.0000.0000.0000)	-
2^{50} (=0x0004.0000.0000.0000)	-
2^{51} (=0x0008.0000.0000.0000)	-
2^{52} (=0x0010.0000.0000.0000)	-
2^{53} (=0x0020.0000.0000.0000)	-
2^{54} (=0x0040.0000.0000.0000)	-
2^{55} (=0x0080.0000.0000.0000)	-
2^{56} (=0x0100.0000.0000.0000)	-
2^{57} (=0x0200.0000.0000.0000)	-
2^{58} (=0x0400.0000.0000.0000)	-
2^{59} (=0x0800.0000.0000.0000)	-
2^{60} (=0x1000.0000.0000.0000)	-

21 jsonSink

21.1 Description

The jsonSink plugin generates JSON output in a file `PREFIX_flows.json`, where `PREFIX` is provided via Tranalyzer `-w` or `-W` option.

21.2 Dependencies

21.2.1 External Libraries

If gzip compression is activated (`GZ_COMPRESS=1`), then **zlib** must be installed.

Kali/Ubuntu: `sudo apt-get install zlib1g-dev`

Arch: `sudo pacman -S zlib`

Fedora/Red Hat: `sudo yum install zlib-devel`

Gentoo: `sudo emerge zlib`

OpenSUSE: `sudo zypper install zlib-devel`

Mac OS X: `brew install zlib`[5]

21.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description	Flags
SOCKET_ON	0	Whether to output to a socket (1) or to a file (0)	
SOCKET_ADDR	"127.0.0.1"	Address of the socket	SOCKET_ON=1
SOCKET_PORT	5000	Port of the socket	SOCKET_ON=1
GZ_COMPRESS	0	Compress (gzip) the output	
JSON_SPLIT	1	Split the output file (Tranalyzer -W option)	SOCKET_ON=0
JSON_ROOT_NODE	0	Add a root node (array)	
SUPPRESS_EMPTY_ARRAY	1	Do not output empty fields	
JSON_NO_SPACES	1	Suppress unnecessary spaces	
JS_BUFFER_SIZE	1024*1024	Size of output buffer	
JSON_SUFFIX	"_flows.json"	Suffix for output file	SOCKET_ON=0

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

21.4 Custom File Output

- `PREFIX_flows.json`: JSON representation of Tranalyzer output

21.5 Example

To send compressed data over a socket (`SOCKET_ON=1` and `GZ_COMPRESS=1`):

1. `nc -l 127.0.0.1 5000 | gunzip`
2. `tranalyzer -r file.pcap`

22 lldpDecode

22.1 Description

The lldpDecode plugin analyzes LLDP traffic.

22.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
LLDP_TTL_AGGR	1	Whether (1) or not (0) to aggregate TTL values
LLDP_NUM_TTL	8	Number of different TTL values to store
LLDP_OPT_TLV	1	Whether or not to output optional TLVs info
LLDP_STRLEN	512	Maximum length of strings to store

22.3 Flow File Output

The lldpDecode plugin outputs the following columns:

Column	Type	Description	Flags
lldpStat	H16	Status	
lldpChassis	SC	Chassis ID	
lldpPort	S	Port ID	
lldpTTL	RU16	Time To Live (sec)	
lldpPortDesc	S	Port description	LLDP_OPT_TLV=1
lldpSysName	S	System name	LLDP_OPT_TLV=1
lldpSysDesc	S	System description	LLDP_OPT_TLV=1
lldpCaps_Enabled	H16_H16	Supported and enabled capabilities	LLDP_OPT_TLV=1
lldpMngmtAddr	SC	Management address	LLDP_OPT_TLV=1

22.3.1 lldpStat

The lldpStat column is to be interpreted as follows:

lldpStat	Description
0x0001	Flow is LLDP
0x0002	Mandatory TLV missing
0x0004	Optional TLVs present
0x0008	Reserved TLV type used
0x0010	Organization specific TLV used
0x0020	Unhandled TLV used
0x2000	String truncated... increase LLDP_STRLEN
0x4000	Too many TTL... increase LLDP_NUM_TTL
0x8000	Snapped payload

22.3.2 lldpCaps

The `lldpCaps_Enabled` column is to be interpreted as follows:

lldpCaps	Description
0x0001	Other
0x0002	Repeater
0x0004	Bridge
0x0008	WLAN access point
0x0010	Router
0x0020	Telephone
0x0040	DOCSIS cable device
0x0080	Station only
0x0100-0x8000	Reserved

22.4 Plugin Report Output

The following information is reported:

- Number of LLDP packets

23 macRecorder

23.1 Description

The macRecorder plugin provides the source- and destination MAC address as well as the number of packets detected in the flow separated by an underscore. If there is more than one combination of MAC addresses, e.g., due to load balancing or router misconfiguration, the plugin prints all recognized MAC addresses separated by semicolons. The number of distinct source- and destination MAC addresses can be output by activating the `MR_NPAIRS` flag. The `MR_MANUF` flags controls the output of the manufacturers for the source and destination addresses. The representation of MAC addresses can be altered using the `MR_MAC_FMT` flag.

23.2 Dependencies

23.2.1 Required Files

The file `manuf.txt` is required if `MR_MANUF > 0` and file `maclbl.txt` is required if `MR_MACLBL > 0`.

23.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
`MR_MAC_FMT`	1	Format for MAC addresses. 0: hex, 1: mac, 2: int
`MR_NPAIRS`	1	Whether (1) or not (0) to report number of distinct pairs
`MR_MANUF`	1	0: no manufacturers, 1: short names, 2: long names
`MR_MACLBL`	0	0: no mac label, 1: mac labeling
`MR_MAX_MAC`	16	max number of output MAC address per flow

23.4 Flow File Output

The macRecorder plugin outputs the following columns:

Column	Type	Description	Flags
`macPairs`	U32	Number of distinct src/dst MAC addresses pairs	`MR_NPAIRS=1`
`srcMac_dstMac_numP`	H64_H64_U64	Src/Dst MAC addresses, number of packets	`MR_MAC_FMT=0`
`srcMac_dstMac_numP`	MAC_MAC_U64	Src/Dst MAC addresses, number of packets	`MR_MAC_FMT=1`
`srcMac_dstMac_numP`	U64_U64_U64	Src/Dst MAC addresses, number of packets	`MR_MAC_FMT=2`
`srcManuf_dstManuf`	SC_SC	Src/Dst MAC manufacturers	`MR_MANUF=1`
`srcManuf_dstManuf`	S_S	Src/Dst MAC manufacturers	`MR_MANUF=2`
`srcLbl_dstLbl`	S_S	Src/Dst MAC label	`MR_MACLBL>0`

23.5 Packet File Output

In packet mode (`-s` option), the macRecorder plugin outputs the following columns:

Column	Description	Flags

Column	Description	Flags
srcManuf	Source MAC manufacturer	MR_MANUF=1
dstManuf	Destination MAC manufacturer	MR_MANUF=1

23.6 Example Output

Consider a host with MAC address aa:aa:aa:aa:aa:aa in a local network requesting a website from a public server. Due to load balancing, the opposite flow can be split and transmitted via two routers with MAC addresses bb:bb:bb:bb:bb:bb and cc:cc:cc:cc:cc:cc. The macRecorder plugin then produces the following output:

```
bb:bb:bb:bb:bb:bb_aa:aa:aa:aa:aa:aa_667;cc:cc:cc:cc:cc:cc_aa:aa:aa:aa:aa:aa_666
```

24 modbus

24.1 Description

The modbus plugin analyzes Modbus traffic.

24.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
MB_DEBUG	0	Whether (1) or not (0) to activate debug output
MB_FE_FRMT	0	Function/Exception codes representation: 0: hex, 1: int
MB_NUM_FUNC	0	Number of function codes to store (0 to hide modbusFC)
MB_UNIQ_FUNC	0	Whether or not to aggregate multiply defined function codes
MB_NUM_FEX	0	Number of function codes causing exceptions to store (0 to hide modbusFEx)
MB_UNIQ_FEX	0	Whether or not to aggregate multiply defined function codes causing exceptions
MB_NUM_EX	0	Number of exception codes to store (0 to hide modbusExC)
MB_UNIQ_EX	0	Whether or not to aggregate multiply defined exception codes

24.3 Flow File Output

The modbus plugin outputs the following columns:

Column	Type	Description	Flags
modbusStat	H16	Status	
modbusUID	U8	Unit identifier	
modbusNPkts	U32	Number of Modbus packets	
modbusNumEx	U16	Number of exceptions	
modbusFCBF	H64	Aggregated function codes	
modbusFC	RH8	List of function codes	MB_NUM_FUNC>0
modbusFExBF	H64	Aggregated function codes which caused exceptions	
modbusFEx	RH8	List of function codes which caused exceptions	MB_NUM_FEX>0
modbusExCBF	H16	Aggregated exception codes	
modbusExC	RH8	List of exception codes	MB_NUM_EX>0

24.3.1 modbusStat

The modbusStat column is to be interpreted as follows:

modbusStat	Description
0x0001	Flow is Modbus

modbusStat	Description
0x0002	Non-modbus protocol identifier
0x0004	Unknown function code
0x0008	Unknown exception code
0x0010	Multiple unit identifiers
0x0100	List of function codes truncated... increase MB_NUM_FUNC
0x0200	List of function codes which caused exceptions truncated... increase MB_NUM_FEX
0x0400	List of exception codes truncated... increase MB_NUM_EX
0x4000	Snapped packet
0x8000	Malformed packet

24.3.2 modbusFC and modbusFCBF

The modbusFC and modbusFCBF columns are to be interpreted as follows:

modbusFC	modbusFCBF	Description
1 = 0x01	0x0000 0000 0000 0002	Read Coils
2 = 0x02	0x0000 0000 0000 0004	Read Discrete Inputs
3 = 0x03	0x0000 0000 0000 0008	Read Multiple Holding Registers
4 = 0x04	0x0000 0000 0000 0010	Read Input Registers
5 = 0x05	0x0000 0000 0000 0020	Write Single Coil
6 = 0x06	0x0000 0000 0000 0040	Write Single Holding Register
7 = 0x07	0x0000 0000 0000 0080	Read Exception Status
8 = 0x08	0x0000 0000 0000 0100	Diagnostic
11 = 0x0b	0x0000 0000 0000 0800	Get Com Event Counter
12 = 0x0c	0x0000 0000 0000 1000	Get Com Event Log
15 = 0x0f	0x0000 0000 0000 8000	Write Multiple Coils
16 = 0x10	0x0000 0000 0001 0000	Write Multiple Holding Registers
17 = 0x11	0x0000 0000 0002 0000	Report Slave ID
20 = 0x14	0x0000 0000 0010 0000	Read File Record
21 = 0x15	0x0000 0000 0020 0000	Write File Record
22 = 0x16	0x0000 0000 0040 0000	Mask Write Register
23 = 0x17	0x0000 0000 0080 0000	Read/Write Multiple Registers
24 = 0x18	0x0000 0000 0100 0000	Read FIFO Queue
43 = 0x2b	0x0000 0800 0000 0000	Read Decide Identification

24.3.3 modbusFEx and modbusFExBF

The modbusFEx and modbusFExBF columns are to be interpreted as modbusFC and modbusFCBF, respectively.

24.3.4 modbusExC and modbusExCBF

The modbusExC and modbusExCBF column are to be interpreted as follows:

modbusExC	modbusExCBF	Description
1 = 0x01	0x0002	Illegal function code

modbusExC	modbusExCBF	Description
2 = 0x02	0x0004	Illegal data address
3 = 0x03	0x0008	Illegal data value
4 = 0x04	0x0010	Slave device failure
5 = 0x05	0x0020	Acknowledge
6 = 0x06	0x0040	Slave device busy
7 = 0x07	0x0080	Negative acknowledge
8 = 0x08	0x0100	Memory parity error
10 = 0x0a	0x0400	Gateway path unavailable
11 = 0x0b	0x0800	Gateway target device failed to respond

24.4 Packet File Output

In packet mode (`-s` option), the modbus plugin outputs the following columns:

Column	Type	Description	Flags
mbTranId	U16	Transaction Identifier	
mbProtId	U16	Protocol Identifier	
mbLen	U16	Length	
mbUnitId	U8	Unit identifier	
mbFuncCode	H8	Function code	MB_FE_FRMT=0
mbFuncCode	U8	Function code	MB_FE_FRMT=1

24.4.1 mbFuncCode

If `mbFuncCode` column is to be interpreted as follows:

mbFuncCode	Description
$<$ 128 (=0x80)	refer to modbusFC and modbusFCBF
$\geq$ 128 (=0x80)	subtract 128 (=0x80) and refer to modbusFEx and modbusFExBF

24.5 Plugin Report Output

The number of Modbus packets is reported.

25 mongoSink

25.1 Description

The mongoSink plugin outputs flow files to MongoDB.

25.2 Dependencies

25.2.1 External Libraries

This plugin depends on the **libmongoc** library.

Ubuntu: `sudo apt-get install libmongoc-dev`

Arch: `sudo pacman -S mongo-c-driver`

Mac OS X: `brew install mongo-c-driver`

25.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
MONGO_QRY_LEN	2048	Max length for query
MONGO_HOST	"127.0.0.1"	Address of the database
MONGO_PORT	"27017"	Port the database is listening to
MONGO_DBNAME	"tranalyzer"	Name of the database
MONGO_TABLE_NAME	"flow"	Name of the database flow table
MONGO_NUM_DOCS	1	Number of documents (flows) to write in bulk
BSON_SUPPRESS_EMPTY_ARRAY	1	Whether or not to output empty fields
BSON_DEBUG	0	Print debug messages

26 mysqlSink

26.1 Description

The mysqlSink plugin outputs flow files to MySQL database.

26.2 Dependencies

26.2.1 External Libraries

This plugin depends on the **MySQL** library.

Ubuntu: `sudo apt-get install libmysqlclient-dev`

Mac OS X: `brew install mysql-connector-c`

26.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
MYSQL_OVERWRITE_DB	2	0: abort if DB already exists 1: overwrite DB if it already exists 2: reuse DB if it already exists
MYSQL_OVERWRITE_TABLE	2	0: abort if table already exists 1: overwrite table if it already exists 2: append to table if it already exists
MYSQL_TRANSACTION_NFLOWS	40000	0: one transaction > 0: one transaction every *n* flows
MYSQL_QRY_LEN	32768	Max length for query
MYSQL_HOST	"127.0.0.1"	Address of the database
MYSQL_DBPORT	3306	Port the DB is listening to
MYSQL_USER	"mysql"	Username to connect to DB
MYSQL_PASS	"mysql"	Password to connect to DB
MYSQL_DBNAME	"tranalyzer"	Name of the database
MYSQL_TABLE_NAME	"flow"	Name of the table

27 nDPI

27.1 Description

This plugin is a simple wrapper around the nDPI library: https://github.com/ntop/nDPI. It classifies flows according to their protocol/application by analyzing the payload content instead of using the destination port. This plugin produces output to the flow file and to a protocol statistics file. Configuration is achieved by user defined compiler switches in `src/nDPI.h`.

27.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Variable	Default	Description
NDPI_OUTPUT_NUM	0	Whether (1) or not (0) to output a numerical classification.
NDPI_OUTPUT_STR	1	Whether (1) or not (0) to output a textual classification.
NDPI_OUTPUT_STATS	1	Whether (1) or not (0) to output nDPI protocol distribution in a separate file.
NDPI_GUESS_UNKNOWN	1	Whether (1) or not (0) to try guessing unknown protocols.

27.3 Flow File Output

The nDPI plugin outputs the following columns:

Column	Type	Description	Flags
nDPIMasterProto	U16	numerical nDPI master protocol	NDPI_OUTPUT_NUM=1
nDPISubProto	U16	numerical nDPI sub protocol	NDPI_OUTPUT_NUM=1
nDPIclass	S	nDPI based protocol classification	NDPI_OUTPUT_STR=1

27.4 nDPI Numerical Protocol Classification

0 Unknown
1 FTP_CONTROL
2 POP3
3 SMTP
4 IMAP
5 DNS
6 IPP
7 HTTP
8 MDNS
9 NTP
10 NetBIOS
11 NFS
12 SSDP
13 BGP
14 SNMP
15 XDMCP
16 SMBv1
17 Syslog
18 DHCP
19 PostgreSQL
20 MySQL
21 Hotmail
22 Direct_Download_Link
23 POPS
24 AppleJuice
25 DirectConnect
26 ntop
27 COAP
28 VMware
29 SMTPS
30 FacebookZero

31 UBNTAC2
32 Kontiki
33 OpenFT
34 FastTrack
35 Gnutella
36 eDonkey
37 BitTorrent
38 SkypeCall
39 Signal
40 Memcached
41 SMBv23
42 Mining
43 NestLogSink
44 Modbus
45 Free
46 Free
47 Xbox
48 QQ
49 Free_49
50 RTSP
51 IMAPS
52 IceCast
53 PPLive
54 PPStream
55 Zattoo
56 ShoutCast
57 Sopcast
58 Tvants
59 TVUplayer
60 HTTP_Download
61 QQLive
62 Thunder
63 Soulseek
64 SSL_No_Cert
65 IRC
66 Ayiya
67 Unencrypted_Jabber
68 MSN
69 Oscar
70 Yahoo
71 BattleField
72 GooglePlus
73 VRRP
74 Steam
75 HalfLife2
76 WorldOfWarcraft
77 Telnet
78 STUN
79 IPsec
80 GRE
81 ICMP
82 IGMP
83 EGP
84 SCTP
85 OSPF
86 IP_in_IP
87 RTP
88 RDP
89 VNC
90 PcAnywhere
91 SSL
92 SSH
93 Usenet
94 MGCP
95 IAX
96 TFTP
97 AFP
98 Stealthnet
99 Aimini
100 SIP
101 TruPhone
102 ICMPV6
103 DHCPV6
104 Armagetron
105 Crossfire
106 Dofus
107 Fiesta
108 Florensia
109 Guildwars
110 HTTP_ActiveSync
111 Kerberos
112 LDAP
113 MapleStory
114 MsSQL-TDS
115 PPTP
116 Warcraft3
117 WorldOfKungFu
118 Slack
119 Facebook
120 Twitter

121 Dropbox
122 GMail
123 GoogleMaps
124 YouTube
125 Skype
126 Google
127 DCE_RPC
128 NetFlow
129 sFlow
130 HTTP_Connect
131 HTTP_Proxy
132 Citrix
133 NetFlix
134 LastFM
135 Waze
136 YouTubeUpload
137 GenericProtocol
138 CHECKMK
139 AJP
140 Apple
141 Webex
142 WhatsApp
143 AppleiCloud
144 Viber
145 AppleiTunes
146 Radius
147 WindowsUpdate
148 TeamViewer
149 Tuenti
150 LotusNotes
151 SAP
152 GTP
153 UPnP
154 LLMNR
155 RemoteScan
156 Spotify
157 Messenger
158 H323
159 OpenVPN
160 NOE
161 CiscoVPN
162 TeamSpeak
163 Tor
164 CiscoSkinny
165 RTCP
166 RSYNC
167 Oracle
168 Corba
169 UbuntuONE
170 Whois-DAS
171 Collectd
172 SOCKS
173 Nintendo
174 RTMP
175 FTP_DATA
176 Wikipedia
177 ZeroMQ
178 Amazon
179 eBay
180 CNN
181 Megaco
182 Redis
183 Pando_Media_Booster
184 VHUA
185 Telegram
186 Vevo
187 Pandora
188 QUIC
189 WhatsAppVoice
190 EAQ
191 Ookla
192 AMQP
193 KakaoTalk
194 KakaoTalk_Voice
195 Twitch
196 Free
197 WeChat
198 MPEG_TS
199 Snapchat
200 Sina(Weibo)
201 GoogleHangout
202 IFLIX
203 Github
204 BJNP
205 Free
206 PPStream
207 SMPP
208 DNScrypt
209 TINC
210 Deezer

211 Instagram
212 Microsoft
213 Starcraft
214 Teredo
215 HotspotShield
216 HEP
217 GoogleDrive
218 OCS
219 Office365
220 Cloudflare
221 MS_OneDrive
222 MQTT
223 RX
224 AppleStore
225 OpenDNS
226 Git
227 DRDA
228 PlayStore
229 SOMEIP
230 FIX
231 Playstation
232 Pastebin
233 LinkedIn
234 SoundCloud
235 CSGO
236 LISP
237 Diameter
238 ApplePush
239 GoogleServices
240 AmazonVideo
241 GoogleDocs
242 WhatsAppFiles

27.5 Plugin Report Output

The following information is reported:

- Number of flows classified

27.6 Additional Output

If `NDPI_OUTPUT_STATS=1` then nDPI protocol distribution statistics are output in `PREFIX_nDPI.txt`.

27.7 Post-Processing

The `protStat` script can be used to sort the `PREFIX_nDPI.txt` file for the most or least occurring protocols (in terms of number of packets or bytes). It can output the top or bottom *N* protocols or only those with at least a given percentage:

- list all the options: `protStat --help`
- sorted list of protocols (by packets): `protStat PREFIX_nDPI.txt`
- sorted list of protocols (by bytes): `protStat PREFIX_nDPI.txt -b`
- top 10 protocols (by packets): `protStat PREFIX_nDPI.txt -n 10`
- bottom 5 protocols (by bytes): `protStat PREFIX_nDPI.txt -n -5 -b`
- protocols with packets percentage greater than 20%: `protStat PREFIX_nDPI.txt -p 20`
- protocols with bytes percentage smaller than 5%: `protStat PREFIX_nDPI.txt -b -p -5`

27.8 How to Update nDPI to New Version

- download latest stable version (or git clone and checkout stable branch)
- delete `src/nDPI` and replace it with this new version
- run the `./new_ndpi_prepatch.sh` script
- `cd src/nDPI/`
- edit `configure.ac`

```
--- configure.ac.origin
+++ configure.ac
@@ -119,9 +119,9 @@

 dnl> https://github.com/json-c/json-c
 AC_ARG_ENABLE([json-c],
-    AS_HELP_STRING([--disable-json-c], [Disable json-c support]))
+    AS_HELP_STRING([--enable-json-c], [Enable json-c support]))

-AS_IF([test "x$enable_json_c" != "xno"], [
+AS_IF([test "x$enable_json_c" = "xyes"], [
        PKG_CONFIG_PATH=/usr/local/share/pkgconfig:$PKG_CONFIG_PATH
        pkg-config --exists json-c
        AS_IF([test "$?" == "0"],
@@ -147,7 +147,7 @@

 AC_CHECK_LIB(pthread, pthread_setaffinity_np, AC_DEFINE_UNQUOTED(HAVE_PTHREAD_SETAFFINITY_NP
     , 1, [libc has pthread_setaffinity_np]))

-AC_CONFIG_FILES([Makefile example/Makefile example/Makefile.dpdk tests/Makefile libndpi.pc
    src/include/ndpi_define.h src/lib/Makefile])
+AC_CONFIG_FILES([Makefile libndpi.pc src/include/ndpi_define.h src/lib/Makefile])
 AC_CONFIG_HEADERS(src/include/ndpi_config.h)
 AC_SUBST(GIT_RELEASE)
 AC_SUBST(NDPI_MAJOR)
```

- edit `Makefile.am`

```
--- Makefile.am.origin
+++ Makefile.am
@@ -1,5 +1,5 @@
 ACLOCAL_AMFLAGS = -I m4
-SUBDIRS = src/lib example tests
+SUBDIRS = src/lib

 pkgconfigdir = $(prefix)/libdata/pkgconfig
 pkgconfig_DATA = libndpi.pc
```

- Replace the `proto.tex` file using the `prototex` utiliy and regenerate doc.
- Add the new files to SVN and delete removed files before commit.

28 netflowSink

28.1 Description

This plugin is an interface of Tranalyzer to a netflow collector.

28.2 Dependencies

28.2.1 Other Plugins

This plugin requires the basicFlow , basicStats and tcpFlags plugins.
In addition, the macRecorder plugin is recommended, but optional.

28.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
NF_SERVADD	"127.0.0.1"	Destination address
NF_DPORT	9995	Destination port
NF_SOCKTYP	0	Socket type: 0: UDP; 1: TCP
NF_VER	9	Netflow version 9 or 10
NF_NUM4FLWS	200	Max # of IPv4 flows in one message
NF_NUM6FLWS	100	Max # of IPv6 flows in one message

28.4 Example

To collect T2 flow data with **nfcapd**, use the following command:

```
nfcapd -T all -B 1000000 -n wurst,127.0.0.1,.
```

29 nFrstPkts

29.1 Description

The nFrstPkts plugin supplies the Packet Length (PL) and Interarrival Times (IAT) of the *N* first packets per flow as a column. The default value for *N* is 20. It complements the packet mode (`-s` option) with flow based view for the *N* first packets signal. The plugin supplies several configuration options of how the resulting packet length signal should be represented. Using the `fpsGplt` script files are generated readily post processable by any command line tool (AWK, Perl), Excel or Data mining suit, such as SPSS. As outlined in the configuration below, Signals can be produced with IAT, or relative/absolute time. Also the aggregation of bursts into a single pulse can be configured via `NFRST_MINIAT`. `NFRST_MINPLAVE` controls the meaning of the PL value in puls aggregation mode. If 0 it corresponds to the BPP measure currently used in research for categorizing media content.

29.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description	Flags
`NFRST_IAT`	1	0: Time releative to flow start, 1: Interarrival Time, 2: Absolute Time	
`NFRST_BCORR`	0	0: A,B start at 0.0 1: B shift by flow start	`NFRST_MINIATS=0`
`NFRST_MINIATS`	0	0: Standard IAT Sequence 1: Minimal Pkt IAT s defining a pulse signal	
`NFRST_MINIATU`	0	0: Standard IAT Sequence, 1: Minimal Pkt IAT us defining a pulse signal	
`NFRST_MINPLENFRC`	2	Minimal pulse length fraction	
`NFRST_PLAVE`	1	0: Sum PL (BPP), 1: Average PL	`NFRST_MINIATS>0||` `NFRST_MINIATU>0`
`NFRST_PKTCNT`	20	Number of packets to record	
`NFRST_HDRINFO`	0	add L3,L4 Header length	
`NFRST_XCLD`	0	0: include all, 1: include `[NFRST_XMIN,NFRST_XMAX]`	
`NFRST_XMIN`	1	min PL boundary	`NFRST_XCLD=1`
`NFRST_XMAX`	`UINT16_MAX`	max PL boundary	`NFRST_XCLD=1`

For the rest of this document, `NFRST_MINIAT` is used to represent `(NFRST_MINIATS>0||NFRST_MINIATU>0)`.

29.3 Flow File Output

The nFrstPkts plugin outputs the following columns:

Column	Type	Description	Flags
`nFpCnt`	U32	Number of signal samples	
`L2L3L4Pl_Iat`	R(U16_UT)	L2/L3/L4 or payload length and inter-arrival times for the N first packets	`NFRST_HDRINFO=0&&` `NFRST_MINIAT=0`

Column	Type	Description	Flags
L2L3L4Pl_Iat_nP	R(U16_UT_UT)	L2/L3/L4 or payload length, inter-arrival times and pulse length for the N first packets	NFRST_HDRINFO=0&& NFRST_MINIAT>0
HD3l_HD4l_ L2L3L4Pl_Iat	R(U8_U8_ _U16_UT)	L3Hdr, L4Hdr, L2/L3/L4 or payload length and inter-arrival times for the N first packets	NFRST_HDRINFO=1&& NFRST_MINIAT=0
HD3l_HD4l_ L2L3L4Pl_Iat_nP	R(U8_U8_U16_ UT_UT)	L3Hdr, L4Hdr, L2/L3/L4 or payload length and inter-arrival times for the N first packets	NFRST_HDRINFO=1&& NFRST_MINIAT>0

29.4 Post-Processing

By invoking the script `fpsGplt` under *trunk/scripts* files are generated for the packet signal in a Gnuplot/Excel/SPSS readable column oriented format. It produces several signal veriants which also can be used for signal processing and AI applications. S. traffic mining tutorial on our webpage

```
>fpsGplt -h
Usage:
    fpsGplt [OPTION...] <FILE>

Optional arguments:

    -f              Flow index to extract, default: all flows
    -d              Flow Direction: 0, 1; default both
    -t              noTime: counts on x axis; default time on x axis
    -i              invert B Flow PL
    -s              time sorted

    -h, --help      Show this help, then exit
```

30 ntpDecode

30.1 Description

The ntpDecode plugin produces a flow based view of NTP operations between computers for anomaly detection and troubleshooting.

30.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
NTP_TS	1	1: print NTP time stamps, 0: no time stamps
NTP_LIVM_HEX	0	Leap indicator, version and mode: 0: split into three values, 1: aggregated hex number

30.3 Flow File Output

The ntpDecode plugin outputs the following columns:

Name	Type	Description	Flags
ntpStat	H8	NTP status, warnings and errors	
ntpLiVM	H8	NTP leap indicator, version number and mode	NTP_LIVM_HEX=1
ntpLi_V_M	U8_U8_U8	NTP leap indicator, version number and mode	NTP_LIVM_HEX=0
ntpStrat	H8	NTP stratum	
ntpRefClkId	IP4	NTP root reference clock ID (stratum ≥ 2)	
ntpRefStrId	SC	NTP root reference string (stratum ≤ 1)	
ntpPollInt	U32	NTP poll interval	
ntpPrec	F	NTP precision	
ntpRtDelMin	F	NTP root delay minimum	
ntpRtDelMax	F	NTP root delay maximum	
ntpRtDispMin	F	NTP root dispersion minimum	
ntpRtDispMax	F	NTP root dispersion maximum	
ntpRefTS	TS	NTP reference timestamp	NTP_TS=1
ntpOrigTS	TS	NTP originate timestamp	NTP_TS=1
ntpRecTS	TS	NTP receive timestamp	NTP_TS=1
ntpTranTS	TS	NTP transmit timestamp	NTP_TS=1

30.3.1 ntpStat

The ntpStat column is to be interpreted as follows:

ntpStat	Description
2^0 (=0x01)	NTP port detected

30.3.2 ntpLiVM

The `ntpLiVM` column is to be interpreted as follows (refer to Section 30.4 for some examples):

ntpLiVM		Description
`xx..`	`....`	Leap indicator
`..xx`	`x...`	Version number
`....`	`.xxx`	Mode

The `Leap Indicator` bits are to be interpreted as follows:

Leap Indicator	Description
`0x0`	No warning
`0x1`	Last minute has 61 seconds
`0x2`	Last minute has 59 seconds
`0x3`	Alarm condition, clock not synchronized

The `Mode` bits are to be interpreted as follows:

Mode	Description
`0x0`	Reserved
`0x1`	Symmetric active
`0x2`	Symmetric passive
`0x3`	Client
`0x4`	Server
`0x5`	Broadcast
`0x6`	NTP control message
`0x7`	Private use

30.3.3 ntpStrat

The `ntpStrat` column is to be interpreted as follows:

ntpStrat	Description
`0x00`	Unspecified
`0x01`	Primary reference
`0x02-0xff`	Secondary reference

30.3.4 ntpRefStrId

The interpretation of the `ntpRefStrId` column depends on the value of ntpStrat. The following table lists some suggested identifiers:

ntpStrat	ntpRefStrId	Description
0x00	DCN	DCN routing protocol
0x00	NIST	NIST public modem
0x00	TSP	TSP time protocol
0x00	DTS	Digital Time Service
0x01	ATOM	Atomic clock (calibrated)
0x01	VLF	VLF radio
0x01	callsign	Generic radio
0x01	LORC	LORAN-C
0x01	GOES	GOES UHF environment satellite
0x01	GPS	GPS UHF positioning satellite

30.4 Examples

- Extract the NTP leap indicator:
  `tawk 'NR > 1 { print rshift(and(strtonum($ntpLiVM), 0xc0), 6) }' out_flows.txt`
- Extract the NTP version:
  `tawk 'NR > 1 { print rshift(and(strtonum($ntpLiVM), 0x38), 3) }' out_flows.txt`
- Extract the NTP mode:
  `tawk 'NR > 1 { printf "%#x\n", and(strtonum($ntpLiVM), 0x7) }' out_flows.txt`

31 ospfDecode

31.1 Description

This plugin analyzes OSPF traffic and provides absolute and relative statistics to the `PREFIX_ospfStats.txt` file. In addition, the `rospf` script extracts the areas, networks and netmasks, along with the routers and their interfaces (Section 31.5).

31.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
`OSPF_OUTPUT_DBD`	0	Output routing tables
`OSPF_OUTPUT_MSG`	0	Output all messages
`OSPF_MASK_AS_IP`	0	How to display netmasks: 0: hex, 1: IP
`OSPF_AREA_AS_IP`	0	How to display areas: 0: int, 1: IP, 2: hex

31.3 Flow File Output

The ospfDecode plugin outputs the following columns:

Column	Type	Description
`ospfStat`	H8	Status
`ospfType`	H8	Message type
`ospfAuType`	H16	Authentication type
`ospfAuPass`	RS	Authentication password (if `ospfAuType == 0x4`)
`ospfArea`	U32/H32	Area ID (see `OSPF_AREA_AS_IP` in Section 31.2)

31.3.1 ospfStat

The hex based status variable (`ospfStat`) is defined as follows:

ospfStat	Description
2^0 (=`0x01`)	OSPF message had invalid TTL ($\neq 1$)
2^1 (=`0x02`)	OSPF message had invalid destination
2^2 (=`0x04`)	OSPF message had invalid type
2^3 (=`0x08`)	OSPF message had invalid checksum
2^4 (=`0x10`)	OSPF message was malformed

The invalid checksum status `0x08` is currently not implemented.
The malformed status `0x10` is currently used to report cases such as possible covert channels, e.g., `authfield` used when `auType` was `NULL`.

31.3.2 ospfType

The hex based message type variable `ospfType` is defined as follows:

ospfType	Description
2^1 (=0x02)	Hello
2^2 (=0x04)	Database Description
2^3 (=0x08)	Link State Request
2^4 (=0x10)	Link State Update
2^5 (=0x20)	Link State Acknowledgement

31.3.3 ospfAuType

The hex based authentication type variable `ospfAuType` is defined as follows:

ospfAuType	Description
2^1 (=0x0002)	Null authentication
2^2 (=0x0004)	Simple password
2^3 (=0x0008)	Cryptographic authentication

31.4 Additional Output

- `PREFIX_ospfStats.txt`: global statistics about OSPF traffic
- `PREFIX_ospfHello.txt` Hello messages (see Section 31.5)
- `PREFIX_ospfDBD.txt`: Routing tables (see Section 31.2)
- `PREFIX_ospfMsg.txt`: All other messages (see Section 31.2)

31.5 Post-Processing

31.5.1 rospf

Hello messages can be used to discover the network topology and are stored in the `PREFIX_ospfHello.txt` file. The script `rospf` extracts the areas, networks, netmasks, routers and their interfaces:

```
./scripts/rospf PREFIX_ospfHello.txt
```

31.5.2 dbd

If `OSPF_OUTPUT_DBD` is activated (Section 31.2), database description messages are stored in a file `PREFIX_ospfDBD.txt`. The `dbd` script formats this file to produce an output similar to that of standard routers:

```
./scripts/dbd PREFIX_ospfDBD.txt
```

```
Name    Area    Network          Netmask
N1      0       192.168.21.0     0xffffff00
N2      1       192.168.16.0     0xffffff00
N3      1       192.168.22.0     0xfffffffc
...

Router    Interface_n       Network_n
R1        192.168.22.29     N11    192.168.21.4    N5    192.168.22.25    N10
R2        192.168.22.5      N12    192.168.16.1    N0    192.168.22.1     N6
R3        192.168.22.10     N13    192.168.21.2    N5    192.168.22.6     N12
...

Router    Connected Routers
R0        R2    R4    R6    R7    R8
R1        R2    R4
R2        R0    R1    R4    R8
...
```

```
OSPF Router with ID (192.168.22.10)

Router Link States (Area 1)

Link ID          ADV Router       Age    Seq#          Checksum
192.168.22.5     192.168.22.5     4      0x80000002    0x38ce
192.168.22.10    192.168.22.10    837    0x80000002    0x6b0f
192.168.22.9     192.168.22.9     837    0x80000002    0x156c

Net Link States (Area 1)

Link ID          ADV Router       Age    Seq#          Checksum
192.168.22.6     192.168.22.10    4      0x80000001    0x150b
192.168.22.9     192.168.22.9     838    0x80000001    0x39e0

Summary Net Link States (Area 1)

Link ID          ADV Router       Age    Seq#          Checksum
192.168.17.0     192.168.22.9     735    0x80000001    0x5dd9
192.168.17.0     192.168.22.10    736    0x80000001    0x57de
192.168.18.0     192.168.22.9     715    0x80000001    0x52e3
...
```

32 p0f

32.1 Description

The p0f plugin tries to fingerprint OS and applications.

32.2 Dependencies

32.2.1 Other Plugins

This plugin requires the **sslDecode** plugin with the following flags activated, i.e., set to 1:

- `SSL_EXT_LIST`
- `SSL_CIPHER_LIST`

32.2.2 Required Files

The file `p0f-ssl.txt` is required.

32.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
`P0F_SSL_VER`	1	whether or not to consider the version for fingerprint match
`P0F_SSL_NCIPHER`	1	whether or not to consider the number of ciphers for fingerprint match
`P0F_SSL_NUMEXT`	1	whether or not to consider the number of extensions for fingerprint match
`P0F_SSL_FLAGS`	1	whether or not to consider flags for fingerprint match
`P0F_SSL_CIPHER`	1	whether or not to consider ciphers for fingerprint match
`P0F_SSL_EXT`	1	whether or not to consider extensions for fingerprint match
`P0F_SSL_ELEN`	6	Maximum length of cipher or extension
`P0F_SSL_NSIG`	64	Maximum number of signatures to read
`P0F_SSL_SLEN`	128	Maximum length of a string (os, browser, comment)
`P0F_SSL_LLEN`	1024	Maximum length of a line in the DB
`P0F_SSL_DB`	`"p0f-ssl.txt"`	Name of the database to use

32.4 Flow File Output

The p0f plugin outputs the following columns:

Column	Type	Description
`p0fSSLRule`	U16	p0f SSL fingerprint rule number
`p0fSSLOS`	S	p0f SSL OS fingerprint
`p0fSSLOS2`	S	p0f SSL OS fingerprint (2)
`p0fSSLBrowser`	S	p0f SSL browser fingerprint

Column	Type	Description
p0fSSLComment	S	p0f SSL fingerprint comment

32.5 References

- https://idea.popcount.org/2012-06-17-ssl-fingerprinting-for-p0f

33 pcapd

33.1 Description

The pcapd plugin can be used to create PCAP files based on some criteria such as flow indexes (Section 33.3.1) or alarms raised by other plugins (Section Section 33.3.2).

33.2 Dependencies

If `PD_MODE=4`, the libpcap version must be at least 1.7.2. (In this mode, the plugin uses the `pcap_dump_open_append()` function which was introduced in the libpcap in February 12, 2015.)

33.3 Configuration Flags

The following flags can be used to configure the plugin:

Variable	Default	Description	Flags
`PD_MODE_IN`	0	0: extract flows listed in input file (if `-e` option was used), or extract flows if alarm bit is set (if `-e` option was not used) 1: dump all packets	
`PD_EQ`	1	whether to save matching (1) or non-matching (0) flows	`PD_MODE_IN=0`
`PD_MODE_OUT`	0	0: one pcap 1: one pcap per flow	
`PD_SPLIT`	1	Split the output file (Tranalyzer `-W` option)	
`PD_FORMAT`	0	Format of the input file (`-e` option): 0: flow index only, 1: flow file format	
`PD_MAX_FD`	128	Maximum number of simultaneously open file descriptors	`PD_MODE_OUT=1`
`PD_SUFFIX`	"`.pcap`"	pcap file extension	

33.3.1 PD_MODE_IN=0, –e option used

The idea behind this mode (`PD_MODE_IN=0` and Tranalyzer `-e` option used) is to use `awk` to extract flows of interest and then the pcapd plugin to create one or more PCAP with all those flows. The format of the file must be as follows:

`PD_FORMAT=0`	The first column must be the flow index (the rest (optionnal) is ignored): `1234 ...`
`PD_FORMAT=1`	The second column must be the flow index: `A 1234 ...`

Lines starting with `'%'`, `'#'`, a space or a tab are ignored, along with empty lines.

Flows whose index appears in the `-e` file will be dumped in a file named `PREFIX_PD_SUFFIX`, where `PREFIX` is the value given to Tranalyzer `-e` option. Note that if `PD_EQ=0`, then flows whose index does **not** appear in the file will be dumped.

33.3.2 PD_MODE_IN=0, –e option not used

In this mode (`PD_MODE_IN=0` and Tranalyzer `-e` option **NOT** used), every flow whose status bit `FL_ALARM=0x20000000` is set (`PD_EQ=1`) or not set (`PD_EQ=0`) will be dumped in a file named `PREFIX_PD_SUFFIX`, where `PREFIX` is the value given to Tranalyzer `-w` or `-W` option.

33.3.3 PD_MODE_IN=1

In this mode, all the packets are dumped into one or more PCAP files. If Tranalyzer `-W` option is used, then the pcap files will be split accordingly. For example, the following command will create PCAP files of 100MB each: `tranalyzer -i eth0 -W out:100M`

33.3.4 PD_MODE_OUT=1

In this mode, every flow will have its own PCAP file, whose name will end with the flow index.

33.4 Additional Output

A PCAP file with suffix `PD_SUFFIX` will be created. The prefix and location of the file depends on the configuration of the plugin.

- If Tranalyzer `-e` option was used, the file is named according to the `-e` option.
- Otherwise the file is named according to the `-w` or `-W` option.

33.5 Examples

For the following examples, it is assumed that Tranalyzer was run as follows, with the *basicFlow* and *txtSink* plugins in their default configuration:

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

The column numbers can be obtained by looking in the file `out_headers.txt` or by using tawk .

33.5.1 Extracting ICMP Flows

To create a PCAP file containing ICMP flows only, proceed as follows:

1. Identify the *"Layer 4 protocol"* column in `out_headers.txt` (column 14):
   `grep "Layer 4 protocol" out_headers.txt`
2. Extract all flow indexes whose protocol is ICMP (1):
   `awk -F'\t' '$14 == 1 { print $2 }' out_flows.txt > out_icmp.txt`
3. Configure pcapd.h as follows: `PD_MODE_IN=0, PD_EQ=1`
4. Build the pcapd plugin: `cd $T2HOME/pcapd/; ./autogen.sh`
5. Re-run Tranalyzer with the `-e` option:
   `tranalyer -r file.pcap -w out -e out_icmp.txt`
6. The file `out_icmp.txt.pcap` now contains all the ICMP flows.

33.5.2 Extracting Non-ICMP Flows

To create a PCAP file containing non-ICMP flows only, use the same procedure as that of Section 33.5.1, but replace `PD_EQ=1` with `PD_EQ=0` in step 3. Alternatively, replace `$14==1` with `$14!=1` in step 2. Or if an entire flow file is preferred to the flow indexes only, set `PD_FORMAT=1` and replace `print $2` with `print $0` in step 2.

34 pktSIATHisto

34.1 Description

The pktSIATHisto plugin records the PL and IAT of a flow. While the PL reflects the bin, the IAT is divided by default into statistical bins to conserve memory / flow, see example below. Where the low precision is reserved for the most prominent IAT of all known codecs. Nevertheless, it can be configured by the user in any arbitrary way. If the memory is not sufficient then decrease HASHCHAINTABLE_BASE_SIZE in tranalyzer.h.

Bin	Range of IAT(default)
0 – 199	0 ms (incl.) – 200 ms (excl.), partitioned into bins of 1 ms
200 – 239	200 ms (incl.) – 400 ms (excl.), partitioned into bins of 5 ms
240 – 299	400 ms (incl.) – 1 sec. (excl.), partitioned into bins of 10 ms
300	for all IAT higher than 1 sec.

34.2 Configuration Flags

Classifying tasks may require other IAT binning. Then the bin limit `IATBINBu` and the binsize `IATBINWu` constants in *pktSIATHisto.h* need to be adapted as being indicated below using 6 different classes of bins:

```
#define IATSECMAX 6 // max # of section in statistics;
                    // last section comprises all elements > IATBINBu6


#define IATBINBu1   50// bin boundary of section one: [0, 50)ms
#define IATBINBu2   200
#define IATBINBu3   1000
#define IATBINBu4   10000
#define IATBINBu5   100000
#define IATBINBu6 1000000

#define IATBINWu1   10// bin width 1ms
#define IATBINWu2   5
#define IATBINWu3   10
#define IATBINWu4   20
#define IATBINWu5   50
#define IATBINWu6   100

#define IATBINNu1   IATBINBu1 / IATBINWu1// # of bins in section one
#define IATBINNu2   (IATBINBu2 - IATBINBu1) / IATBINWu2 + IATBINNu1
#define IATBINNu3   (IATBINBu3 - IATBINBu2) / IATBINWu3 + IATBINNu2
#define IATBINNu4   (IATBINBu4 - IATBINBu3) / IATBINWu4 + IATBINNu3
#define IATBINNu5   (IATBINBu5 - IATBINBu4) / IATBINWu5 + IATBINNu4
#define IATBINNu6   (IATBINBu6 - IATBINBu5) / IATBINWu6 + IATBINNu5
```

The number of bin sections is defined by `IATSECMAX`, default is 3. The static fields `IATBinBu` and `IATBinWu` need to be adapted when `IATSECMAX` is changed. The static definition in curly brackets of the constant fields `IATBinBu[]`, `IATBinBu[]` and `IATBinBu[]` must adapted as well to the maximal bin size. The constant `IATBINUMAX` including his two dimensional packet length, IAT statistics is being used by the descriptive statistics plugin and can suit as a raw input for subsequent statistical classifiers, such as Bayesian networks or C5.0 trees.

The user is able to customize the output by changing several define statements in the header file *pktSIATHisto.h*. Every change requires a recompilation of the plugin using the *autogen.sh* script.

HISTO_PRINT_BIN == 0, the default case, selects the number of the IAT bin, while 1 supplies the lower bound of the IAT bin's range.
As being outlined in the Descriptive Statistics plugin the output of the plugin can be suppressed by defining PRINT_HISTO to zero.
For specific applications in the AI regime, the distribution can be directed into a separate file if the value PRINT_HISTO_IN_SEPARATE_FILE is different from zero. The suffix for the distribution file is defined by the HISTO_FILE_SUFFIX define.
All switches are listed below:

Name	Default	Description	Flags
HISTO_NODEPOOL_FACTOR	17	multiplication factor redblack tree nodepool	
PRINT_HISTO	1	print histo to flow file	
HISTO_PRINT_BIN	0	Bin number; 0: Minimum of assigned inter arrival time. Example: Bin = 10 -> iat = [50:55) -> min(iat) = 50ms	
HISTO_EARLY_CLEANUP	0	after onFlowTerminate tree information is destroyed. **MUST** be 0 if dependent plugins are loaded	
PSI_XCLD	0	1: include (BS_XMIN,UINT16_MAX]	
PSI_XMIN	1	minimal packet length starts at PSI_XMIN	PSI_XCLD==1
PSI_MOD	0	> 1 : Modulo factor of packet length	
IATSECMAX	3	max # of sections in statistics, last section comprises all elements > IATBINBuN	PSI_XCLD==1

34.3 Flow File Output

The pktSIATHisto plugin outputs the following columns:

Column	Type	Description	Flags
tCnt	U32	Packet size inter-arrival time number of tree entries	
Ps_IatBin_Cnt_PsCnt_IatCnt	R(U16_4xU32)	Packet size inter-arrival time bin histogram	HISTO_PRINT_BIN=0
Ps_Iat_Cnt_PsCnt_IatCnt	R(U16_4xU32)	Packet size min inter-arrival time of bin histo	HISTO_PRINT_BIN=1

All PL-IAT bins greater than zero are appended for each flow in the PREFIX_flows.txt file using the following format:

```
[ps]_[IAT]_[# packets]_[# of packets PL]_[# of packets IAT]
```

the PL-IAT bins are separated by semicolons. The IAT value is the lower bound of the IAT range of a bin.

34.4 Post-Processing

By invoking the script statGplt under *trunk/scripts* files are generated for the 2/3 dim statistics in a Gnuplot/Excel/SPSS column oriented format. The format is:

- For the 3D case: **PL** <tab> **IAT** <tab> **count**
- For the 2D case: **PL** <tab> **count**

34.5 Example Output

Consider a single flow with the following PL and IAT values:

Packet number	PL (bytes)	IAT (ms)	IAT bin
1	50	0	0
2	70	88.2	17
3	70	84.3	16
4	70	92.9	18
5	70	87.1	17
6	60	91.6	18

Packet number two and five have the same PL-IAT combination. Packets number two to five have the same PL and number two and five as well as the number four and six fall within the same IAT bin. Therefore the following sequence is generated:

```
50_0_1_1_1 ; 60_90_1_1_2 ; 70_80_1_4_1 ; 70_85_2_4_2 ; 70_90_1_4_2
```

Note that for better readability spaces are inserted around the semicolons which will not exist in the text based flow file!

35 popDecode

35.1 Description

The popDecode plugin processes MAIL header and content information of a flow. The idea is to identify certain pop mail features and save content. User defined compiler switches are in *popDecode.h*.

35.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
POP_SAVE	0	save content to POP_F_PATH
MXNMLN	21	maximal name length
MXUNM	5	maximal number of users
MXPNM	5	maximal number of passwords/parameters
MXCNM	5	maximal number of content

35.3 Flow File Output

The popDecode plugin outputs the following columns:

Column	Type	Description
popStat	H8	Status bit field
popCBF	H16	POP command codes bit field
popCC	RSC	POP Command Codes
popRM	RU16	POP Response Codes
popUsrNum	U8	number of POP Users
popPwNum	U8	number of POP Passwords
popCNum	U8	number of POP parameters
popUsr	RS	POP Users
popPw	RS	POP Passwords
popC	RS	POP Content

35.3.1 popStat

The popStat column describes the errors encountered during the flow lifetime:

popStat	Name	Description
2^0 (=0x01)	POP2_INIT	pop2 port found
2^1 (=0x02)	POP3_INIT	pop3 port found
2^2 (=0x04)	POP_ROK	response +OK
2^3 (=0x08)	POP_RERR	response -ERR
2^4 (=0x10)	POP_DWF	data storage exists, POP_SAVE == 1
2^4 (=0x20)	POP_DTP	data storage in progress, POP_SAVE == 1
2^6 (=0x40)	POP_RNVL	response not valid or data

popStat	Name	Description
2^7 (=0x80)	POP_OVFL	array overflow

35.3.2 popCBF

The popCBF column describes the commands encountered during the flow lifetime:

popCBF	Name	Description
2^0 (=0x0001)	POP_APOP	Login with MD5 signature
2^1 (=0x0002)	POP_AUTH	Authentication request
2^2 (=0x0004)	POP_CAPA	Get a list of capabilities supported by the server
2^3 (=0x0008)	POP_DELE	Mark the message as deleted
2^4 (=0x0010)	POP_LIST	Get a scan listing of one or all messages
2^5 (=0x0020)	POP_NOOP	Return a +OK reply
2^6 (=0x0040)	POP_PASS	Cleartext password entry
2^7 (=0x0080)	POP_QUIT	Exit session. Remove all deleted messages from the server
2^8 (=0x0100)	POP_RETR	Retrieve the message
2^9 (=0x0200)	POP_RSET	Remove the deletion marking from all messages
2^{10} (=0x0400)	POP_STAT	Get the drop listing
2^{11} (=0x0800)	POP_STLS	Begin a TLS negotiation
2^{12} (=0x1000)	POP_TOP	Get the top n lines of the message
2^{13} (=0x2000)	POP_UIDL	Get a unique-id listing for one or all messages
2^{14} (=0x4000)	POP_USER	Mailbox login
2^{15} (=0x8000)	POP_XTND	

35.4 TODO

- IPv6
- fragmentation
- reply address extraction

36 portClassifier

36.1 Description

The portClassifier plugin classifies the flow according to the destination port meaning. It accepts a default port list `portmap.txt`, automatically installed with the plugin.

36.2 Dependencies

36.2.1 Required Files

The file `portmap.txt` is required.

36.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
`PBC_NUM`	1	Print string representation of port classification
`PBC_STR`	1	Print numeric representation of port classification
`PBC_CLASSFILE`	`"portmap.txt"`	input file for the mapping between ports and applications

36.4 Flow File Output

The portClassifier plugin outputs the following columns:

Column	Type	Description	Flags
`dstPortClassN`	U16	Port based classification of the destination port number	`PBC_NUM=0`
`dstPortClass`	SC	Port based classification of the destination port name	`PBC_STR=1`

37 protoStats

37.1 Description

The protoStats plugin provides protocol/port sorted frequency statistics about the observed OSI layer 4 protocols and ports to the file named `PREFIX_protocols`. Protocols numbers are decoded via a `proto.txt` file, automatically installed with the plugin.

37.2 Dependencies

37.2.1 Required Files

The file `proto.txt` is required.

37.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
`ETH_STAT`	1	Output layer 2 statistics
`SCTP_STAT`	0	Output SCTP statistics
`UDPLITE_STAT`	0	Output UDP-Lite statistics

37.4 Flow File Output

None.

37.5 Additional Output

- `PREFIX_protocols.txt`: protocol statistics

37.6 Post-Processing

The `protStat` script can be used to sort the `PREFIX_protocols.txt` file for the most or least occurring protocols (in terms of number of packets or bytes). It can output the top or bottom *N* protocols or only those with at least a given percentage:

- list all the options: `protStat --help`
- 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`

38 psqlSink

38.1 Description

The psqlSink plugin outputs flow files to PostgreSQL database.

38.2 Dependencies

38.2.1 External Libraries

This plugin depends on the **libpq** library.

Ubuntu: `sudo apt-get install libpq-dev`

Arch: `sudo pacman -S postgresql-libs`

Mac OS X: `brew install postgresql`

38.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
PSQL_OVERWRITE_DB	2	0: abort if DB already exists 1: overwrite DB if it already exists 2: reuse DB if it already exists
PSQL_OVERWRITE_TABLE	2	0: abort if table already exists 1: overwrite table if it already exists 2: append to table if it already exists
PSQL_TRANSACTION_NFLOWS	40000	0: one transaction > 0: one transaction every *n* flows
PSQL_QRY_LEN	32768	Max length for query
PSQL_HOST	"127.0.0.1"	Address of the database
PSQL_PORT	5432	Port of the database
PSQL_USER	"postgres"	Username to connect to DB
PSQL_PASS	"postgres"	Password to connect to DB
PSQL_DBNAME	"tranalyzer"	Name of the database
PSQL_TABLE_NAME	"flow"	Name of the table

38.4 Post-Processing

The following queries can be used to analyze bitfields in PostgreSQL:

- Select all A flows:

```
SELECT to_hex("flowStat"::bigint), *
FROM flow
WHERE ("flowStat"::bigint & 1) = 0::bigint
```

- Select all IPv4 flows:
```
SELECT *
FROM flow
WHERE ("flowStat"::bigint & x'4000'::bigint) != 0::bigint
```
- Select all IPv6 flows:
```
SELECT to_hex("flowStat"::bigint), *
FROM flow
WHERE ("flowStat"::bigint & x'8000'::bigint) != 0::bigint
```

39 pwX

39.1 Description

The pwX plugin extracts usernames and passwords from different plaintext protocols. This plugin produces only output to the flow file. Configuration is achieved by user defined compiler switches in `src/pwX.h`.

39.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Variable	Default	Description
PWX_USERNAME	1	Defines if username column is printed.
PWX_PASSWORD	1	Defines if password column is printed.
PWX_FTP	1	Defines if FTP authentication is extracted.
PWX_POP3	1	Defines if POP3 authentication is extracted.
PWX_IMAP	1	Defines if IMAP authentication is extracted.
PWX_SMTP	1	Defines if SMTP authentication is extracted.
PWX_HTTP_BASIC	1	Defines if HTTP Basic Authorization is extracted.
PWX_HTTP_PROXY	1	Defines if HTTP Proxy Authorization is extracted.
PWX_HTTP_GET	1	Defines if HTTP GET authentication is extracted.
PWX_HTTP_POST	1	Defines if HTTP POST authentication is extracted.
PWX_IRC	1	Defines if IRC authentication is extracted.
PWX_TELNET	1	Defines if Telnet authentication is extracted.
PWX_LDAP	1	Defines if LDAP bind request authentication is extracted.
PWX_PAP	1	Defines if Password Authentication Protocol (PAP) is extracted.
PWX_STATUS	1	Whether or not to extract authentication status (success, error, ...).
PWX_DEBUG	0	Whether or not to activate debug output.

39.3 Flow File Output

The pwX plugin outputs the following columns:

Name	Type	Description	Flags
pwxType	U8	Authentication type	
pwxUser	S	Extracted username	PWX_USERNAME != 0
pwxPass	S	Extracted password	PWX_PASSWORD != 0
pwxStatus	U8	Authentication status	PWX_STATUS != 0

39.3.1 pwxType

The `pwxType` column is to be interpreted as follows:

pwxType	Description
0	No password or username extracted
1	FTP authentication
2	POP3 authentication
3	IMAP authentication
4	SMTP authentication
5	HTTP Basic Authorization
6	HTTP Proxy Authorization
7	HTTP GET authentication
8	HTTP POST authentication
9	IRC authentication
10	Telnet authentication
11	LDAP authentication
12	PAP authentication

39.3.2 pwxStatus

The `pwxStatus` column is to be interpreted as follows:

pwxStatus	Description
0	Authentication status is unknown
1	Authentication was successful
2	Authentication failed

39.4 Plugin Report Output

The number of passwords extracted is reported.

40 radiusDecode

40.1 Description

The radiusDecode plugin analyzes RADIUS traffic.

40.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
RADIUS_DBG	0	Whether (1) or not (0) to print debug messages
RADIUS_NAS	1	Whether (1) or not (1) to output NAS info
RADIUS_FRAMED	1	Whether (1) or not (0) to output framed info
RADIUS_TUNNEL	1	Whether (1) or not (0) to output tunnel info
RADIUS_ACCT	1	Whether (1) or not (0) to output accounting info

40.3 Flow File Output

The radiusDecode plugin outputs the following columns:

Column	Type	Description
radiusStat	H8	Status
radiusAxsReq_Acc_Rej_Chal	4xU16	Access-Request/Accept/Reject/Challenge
radiusAccReq_Resp	U16_U16	Accounting-Request/Response
radiusAccStart_Stop	U16_U16	Accounting Start/Stop
radiusUser	S	Username
radiusServiceTyp	U32	Service type
radiusLoginService	U32	Login-Service
radiusVendor	U32	Vendor Id (SMI)

If RADIUS_NAS=1, the following columns are displayed:

radiusNasId	S	NAS Identifier
radiusNasIp	IP4	NAS IP address
radiusNasPort	U32	NAS IP port
radiusNasPortTyp	U32	NAS port type
radiusNasPortId	S	NAS port Id

If RADIUS_FRAMED=1, the following columns are displayed:

radiusFramedIp	IP4	Framed IP address
radiusFramedMask	IP4	Framed IP netmask
radiusFramedProto	U32	Framed protocol
radiusFramedComp	U32	Framed compression
radiusFramedMtu	U32	Framed MTU

Column	Type	Description
If RADIUS_TUNNEL=1, the following columns are displayed:		
radiusTunnel_Medium	U32_U32	Tunnel type and medium type
radiusTunnelCli	S	Tunnel client endpoint
radiusTunnelSrv	S	Tunnel server endpoint
radiusTunnelCliAId	S	Tunnel client authentication Id
radiusTunnelSrvAId	S	Tunnel server authentication Id
radiusTunnelPref	S	Tunnel preference
If RADIUS_ACCT=1, the following columns are displayed:		
radiusAcctSessId	S	Accounting session Id
radiusAcctSessTime	U32	Accounting session time (seconds)
radiusAcctStatTyp	U32	Accounting status type
radiusAcctTerm	U32	Accounting terminate cause
radiusAcctInOct_OutOct	U32_U32	Accounting input/output octets
radiusAcctInPkt_OutPkt	U32_U32	Accounting input/output packets
radiusAcctInGw_OutGw	U32_U32	Accounting input/output gigawords
radiusConnInfo	S	User connection info
radiusFilterId	S	Filter Identifier
radiusCalledId	S	Called Station Identifier
radiusCallingId	S	Calling Station Identifier
radiusReplyMsg	S	Reply message

40.3.1 radiusStat

The radiusStat column is to be interpreted as follows:

radiusStat	Description
2^0 (=0x01)	Flow is RADIUS
2^0 (=0x02)	Authentication and configuration traffic
2^0 (=0x04)	Accounting traffic
2^2 (=0x10)	Connection successful
2^1 (=0x20)	Connection failed
2^7 (=0x80)	Malformed packet

40.3.2 radiusServiceTyp

The radiusServiceTyp column is to be interpreted as follows:

radiusServiceTyp	Description
1	Login
2	Framed
3	Callback Login

radiusServiceTyp	Description
4	Callback Framed
5	Outbound
6	Administrative
7	NAS Prompt
8	Authenticate Only
9	Callback NAS Prompt
10	Call Check
11	Callback Administrative
12	Voice
13	Fax
14	Modem Relay
15	IAPP-Register
16	IAPP-AP-Check
17	Authorize Only
18	Framed-Management
19	Additional-Authorization

40.3.3 radiusLoginService

The `radiusLoginService` column is to be interpreted as follows:

radiusLoginService	Description
0	Telnet
1	Rlogin
2	TCP Clear
3	PortMaster (proprietary)
4	LAT
5	X25-PAD
6	X25-T3POS
7	Unassigned
8	TCP Clear Quiet (suppresses any NAS-generated connect string)

40.3.4 radiusVendor

The `radiusVendor` column represents the SMI Network Management Private Enterprise Codes which can be found at https://www.iana.org/assignments/enterprise-numbers. Alternatively use `grep` on the file `vendor.txt` as follows: `grep id vendor.txt`, where `id` is the actual Id reported by Tranalyzer, e.g., 4874 for Juniper.

40.3.5 radiusNasPortTyp

The `radiusNasPortTyp` column is to be interpreted as follows:

radiusNasPortTyp	Description
0	Async
1	Sync

radiusNasPortTyp	Description
2	ISDN Sync
3	ISDN Async V.120
4	ISDN Async V.110
5	Virtual
6	PIAFS
7	HDLC Clear Channel
8	X.25
9	X.75
10	G.3 Fax
11	SDSL - Symmetric DSL
12	ADSL-CAP - Asymmetric DSL, Carrierless Amplitude Phase Modulation
13	ADSL-DMT - Asymmetric DSL, Discrete Multi-Tone
14	IDSL - ISDN Digital Subscriber Line
15	Ethernet
16	xDSL - Digital Subscriber Line of unknown type
17	Cable
18	Wireless - Other
19	Wireless - IEEE 802.11
20	Token-Ring
21	FDDI
22	Wireless - CDMA2000
23	Wireless - UMTS
24	Wireless - 1X-EV
25	IAPP
26	FTTP - Fiber to the Premises
27	Wireless - IEEE 802.16
28	Wireless - IEEE 802.20
29	Wireless - IEEE 802.22
30	PPPoA - PPP over ATM
31	PPPoEoA - PPP over Ethernet over ATM
32	PPPoEoE - PPP over Ethernet over Ethernet
33	PPPoEoVLAN - PPP over Ethernet over VLAN
34	PPPoEoQinQ - PPP over Ethernet over IEEE 802.1QinQ
35	xPON - Passive Optical Network
36	Wireless - XGP
37	WiMAX Pre-Release 8 IWK Function
38	WIMAX-WIFI-IWK: WiMAX WIFI Interworking
39	WIMAX-SFF: Signaling Forwarding Function for LTE/3GPP2
40	WIMAX-HA-LMA: WiMAX HA and or LMA function
41	WIMAX-DHCP: WiMAX DCHP service
42	WIMAX-LBS: WiMAX location based service
43	WIMAX-WVS: WiMAX voice service

40.3.6 radiusFramedProto

The `radiusFramedProto` column is to be interpreted as follows:

radiusFramedProto	Description
1	PPP
2	SLIP
3	AppleTalk Remote Access Protocol (ARAP)
4	Gandalf proprietary SingleLink/MultiLink protocol
5	Xylogics proprietary IPX/SLIP
6	X.75 Synchronous
7	GPRS PDP Context

40.3.7 radiusFramedComp

The `radiusFramedComp` column is to be interpreted as follows:

radiusFramedComp	Description
0	None
1	VJ TCP/IP header compression
2	IPX header compression
3	Stac-LZS compression

40.3.8 radiusTunnel_Medium

The `radiusTunnel_Medium` column is to be interpreted as follows:

radiusTunnel	Description
1	Point-to-Point Tunneling Protocol (PPTP)
2	Layer Two Forwarding (L2F)
3	Layer Two Tunneling Protocol (L2TP)
4	Ascend Tunnel Management Protocol (ATMP)
5	Virtual Tunneling Protocol (VTP)
6	IP Authentication Header in the Tunnel-mode (AH)
7	IP-in-IP Encapsulation (IP-IP)
8	Minimal IP-in-IP Encapsulation (MIN-IP-IP)
9	IP Encapsulating Security Payload in the Tunnel-mode (ESP)
10	Generic Route Encapsulation (GRE)
11	Bay Dial Virtual Services (DVS)
12	IP-in-IP Tunneling
13	Virtual LANs (VLAN)

radiusMedium	Description
1	IPv4 (IP version 4)
2	IPv6 (IP version 6)
3	NSAP
4	HDLC (8-bit multidrop)
5	BBN 1822

radiusMedium	Description
6	802 (includes all 802 media plus Ethernet “canonical format”)
7	E.163 (POTS)
8	E.164 (SMDS, Frame Relay, ATM)
9	F.69 (Telex)
10	X.121 (X.25, Frame Relay)
11	IPX
12	Appletalk
13	Decnet IV
14	Banyan Vines
15	E.164 with NSAP format subaddress

40.3.9 radiusAcctStatTyp

The `radiusAcctStatTyp` column is to be interpreted as follows:

radiusAcctStatTyp	Description
1	Start
2	Stop
3	Interim-Update
7	Accounting-On
8	Accounting-Off
9	Tunnel-Start
10	Tunnel-Stop
11	Tunnel-Reject
12	Tunnel-Link-Start
13	Tunnel-Link-Stop
14	Tunnel-Link-Reject
15	Failed

40.3.10 radiusAcctTerm

The `radiusAcctTerm` column is to be interpreted as follows:

radiusAcctTerm	Description
1	User Request
2	Lost Carrier
3	Lost Service
4	Idle Timeout
5	Session Timeout
6	Admin Reset
7	Admin Reboot
8	Port Error
9	NAS Error
10	NAS Request
11	NAS Reboot

radiusAcctTerm	Description
12	Port Unneeded
13	Port Preempted
14	Port Suspended
15	Service Unavailable
16	Callback
17	User Error
18	Host Request
19	Supplicant Restart
20	Reauthentication Failure
21	Port Reinitialized
22	Port Administratively Disabled
23	Lost Power

40.4 Plugin Report Output

The number of RADIUS, Access, Access-Accept, Access-Reject and Accounting packets is reported.

40.5 References

- RFC2865: Remote Authentication Dial In User Service (RADIUS)
- RFC2866: RADIUS Accounting
- RFC2867: RADIUS Accounting Modifications for Tunnel Protocol Support
- RFC2868: RADIUS Attributes for Tunnel Protocol Support
- RFC2869: RADIUS Extensions
- https://www.iana.org/assignments/radius-types/radius-types.xhtml

41 regex_pcre

41.1 Description

The regex_pcre plugin provides a full PCRE compatible regex engine.

41.2 Dependencies

41.2.1 External Libraries

This plugin depends on the **pcre** library.

Ubuntu: `sudo apt-get install libpcre3-dev`

OpenSUSE: `sudo zypper install pcre-devel`

Mac OS X: `brew install pcre`

41.2.2 Other Plugins

If `LABELSCANS=1`, then this plugin requires the tcpFlags plugin.

41.2.3 Required Files

The file `regexfile.txt` is required. See Section 41.3.3 for more details.

41.3 Configuration Flags

41.3.1 regfile_pcre.h

The compiler constants in *regfile_pcre.h* control the pre-processing and compilation of the rule sets supplied in the regex file during the initialisation phase of Tranalyzer.

Name	Default	Description
RULE_OPTIMIZE	0	0: No opt rules allocated 1: Allocate opt rule structure & compile regex
REGEX_MODE	PCRE_DOTALL	Regex compile time options

41.3.2 regex_pcre.h

The compiler constants in *regex_pcre.h* control the execution and the output the rule matches.

Variable	Default	Description	Flags
EXPERTMODE	0	0: Alarm with highest severity: class type & severity, 1: full info	
PKTTIME	0	0: no time, 1: timestamp when rule matched	
LABELSCANS	0	0: No scans, 1: label scans (depends on tcpFlags)	
MAXREGPOS	30	Maximal # of matches stored / flow	
OVECCOUNT	1	regex internal: maximal # of regex output vectors	

Variable	Default	Description	Flags
REXPOSIX_FILE	"regexfile.txt"	Name of regex file under *./tranalyzer/plugins*	

41.3.3 regexfile.txt

The *regexfile.txt* file has the following format:

```
# ID Predecessor Flags ANDMask  ANDPin ClassID Severity Sel Dir Proto srcPort   dstPort offset
    Regex
# single rule
1   0   0x80    0x0000  0x0000  15  3   0x8b    0x0001  6   0   80  0   \x6A.{1,}\x6B\x3C\x24\x0B\
    x60\x6A.*
# single rule
3   1   0x80    0x0000  0x0000  15  3   0x82    0x0001  6   0   80  8   \x31\xDB\x8D\x43\x0D\xCD\
    x80\x66.*\x31
# root rules to following tree
202 0   0x11    0x0000  0x0000  20  4   0x41    0x0001  6   0   80  20  ^http
203 0   0x10    0x0000  0x0000  20  4   0x41    0x0001  6   0   80  20  GET
# sucessors and predesessors
204 202 0x01    0x0000  0x0001  43  2   0x85    0x0001  6   0   445 0   Volume Serial Number
204 203 0x40    0x0000  0x0002  40  2   0x8f    0x0001  6   666 666 0   (?i)Command completed(?-i)
# successors 20t5 & 205 to 204 AND ruleset
205 204 0x81    0x0003  0x0000  40  3   0x00    0x0001  0   0   20  0   ^get .*porno.*
206 204 0x80    0x0002  0x0000  35  3   0x00    0x0000  0   0   21  0   ^FTP
```

Lines starting with a '#' denote a comment line and will be ignored. All kind of rule trees can be formed using rules also acting on multiple packets using different ID's and Predecessor as outlined in the example above. Regex rules with the same ID denote combined predecessors to other rules. Default is an OR operation unless ANDPin bits are set. These bits denote the different inputs to a bitwise AND. The output is then provided to the successor rule which compares with the ANDMask bit field whether all necessary rules are matched. Then an evaluation of the successor rule can take place. Thus, arbitrary rule trees can be constructed and results of predecessors can be used for multiple successor rules. The variable Flags controls the basic PCRE rule interpretation and the flow alarm production (see the table below), e.g. only if bit eight is set and alarm flow output is produced. ClassID and Severity denote information being printed in the flow file if the rule fires.

Flags	Description
2^0 (=0x01)	PCRE_CASELESS
2^1 (=0x02)	PCRE_MULTILINE
2^2 (=0x04)	PCRE_DOTALL
2^3 (=0x08)	PCRE_EXTENDED
2^4 (=0x10)	Internal state: successor found
2^5 (=0x20)	Internal state: predecessor matched
2^6 (=0x40)	Preserve alarm in queue for later use
2^7 (=0x80)	Print alarm in flow file

The Sel column controls the header selection of a rule in the lower nibble and the start of regex evaluation in the higher nibble. The position of the bits in the control byte are outlined below:

Sel	Description
2^0 (=0x01)	Activate dir field
2^1 (=0x02)	Activate L4Proto field
2^2 (=0x04)	Activate srcPort field
2^3 (=0x08)	Activate dstPort field
2^4 (=0x10)	Header start: Layer 2
2^5 (=0x20)	Header start: Layer 3
2^6 (=0x40)	Header start: Layer 4
2^7 (=0x80)	Header start: Layer 7

The higher nibble selects which flow direction (A=0 or B=1), protocol, source and destination port will be evaluated per rule, all others will be ignored. The dir field might contain other bits meaning more selection options in future. The offset column depicts the start of the regex evaluation from the selected header start, default value 0. The Regex column accepts a full PCRE regex term. If the regex is not correct, the rule will be discarded displaying an error message in the Tranalyzer report.

41.4 Flow File Output

The regex_pcre plugin outputs the following columns:

Column	Type	Description	Flags
RgxCnt	U16	Regexp match count	
RgxClTyp	U8	Classtype	EXPERTMODE=0
RgxSev	U8	Severity	EXPERTMODE=0
RgxN_B_RID_ Amsk_F_CT_Sv	R(4xU16_) H8_2xU8)	Packet, byte position, regfile ID, AND mask, flags, classtype, severity	EXPERTMODE=1&& PKTTIME=0
RgxT_N_B_RID_ Amsk_F_CT_Sv	R(TS_4xU16_ H8_2xU8)	Time, packet, byte position, regfile ID, AND mask, flags, classtype, severity	EXPERTMODE=1&& PKTTIME=1

41.5 Plugin Report Output

The following information is reported:

- Number of alarms

42 sctpDecode

42.1 Description

The sctpDecode plugin produces a flow based view of SCTP operations between computers for anomaly detection and troubleshooting purposes.

42.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description	Flags
SCTP_CRC32CHK	0	1: CRC32 check	
SCTP_ADL32CHK	0	1: Adler32 check	
SCTP_CHNKVAL	0	1: chunk type value, 0: chunk type field	
SCTP_CHNKSTR	0	1: chunk types as string	SCTP_CHNKVAL=1
SCTP_MAXCTYPE	15	1: maximum chunk types to store/flow	SCTP_CHNKVAL=1

42.3 Flow File Output

The sctpDecode plugin outputs the following columns:

Column	Type	Description	Flags
sctpStat	H8	SCTP status	
sctpNumS	U16	SCTP max Number of streams/stream number	
sctpPID	U32	SCTP Payload ID	
sctpVTag	H32	SCTP verification tag	
sctpTypeBf	H16	SCTP aggregated type bit field	SCTP_CHNKVAL=0
sctpType	H8R	SCTP uniq type value	SCTP_CHNKVAL=1&&SCTP_CHNKSTR=0
sctpTypeN	SCR	SCTP uniq type name	SCTP_CHNKVAL=1&&SCTP_CHNKSTR=1
sctpCntD_I_A	3U16	SCTP Data_Init_Abort count	
sctpCFlgs	H8	SCTP aggregated chunk flag	
sctpCCBF	H16	SCTP aggregated error cause code bit field	
sctpIS	U16	SCTP inbound streams	
sctpOS	U16	SCTP outbound streams	
sctpIARW	U32	SCTP Initial Advertised Receiver Window	
sctpIARWMin	U32	SCTP Initial Advertised Receiver Window Minimum	
sctpIARWMax	U32	SCTP Initial Advertised Receiver Window Maximum	
sctpARW	F	SCTP Advertised Receiver Window	

42.3.1 sctpStat

The sctpStat column is to be interpreted as follows:

sctpStat	Description
2^0 (=0x01)	Adler32 error

sctpStat	Description
2^1 (=0x02)	CRC32 error
2^2 (=0x04)	—
2^3 (=0x08)	Chunk truncated
2^6 (=0x10)	—
2^7 (=0x20)	Type Field overflow
2^4 (=0x40)	Type BF: Do not report
2^5 (=0x80)	Type BF: Stop processing of the packet

42.3.2 sctpCFlgs

The `sctpCFlgs` column is to be interpreted as follows:

sctpCFlgs	Description
2^0 (=0x01)	Last segment
2^1 (=0x02)	First segment
2^2 (=0x04)	Ordered delivery
2^3 (=0x08)	Possibly delay SACK
2^6 (=0x10)	—
2^7 (=0x20)	—
2^4 (=0x40)	—
2^5 (=0x80)	—

42.4 Packet File Output

In packet mode (`-s` option), the sctpDecode plugin outputs the following columns:

Column	Type	Description
`sctpVerifTag`	H32	Verification tag
`sctpChunkType_Sid_Flags_Len`	U8/S_H8_U16(R)	Chunk type, flags and length
`sctpNChunks`	U8	Number of chunks

43 smbDecode

43.1 Description

The smbDecode plugin analyzes SMB2 traffic.

43.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description	Flags
SMB1_DECODE	0	Whether or not to decode SMB1 (beta)	
SMB_SECBLOB	0	Whether or not to decode security blob (beta)	
SMB_NUM_FNAME	5	number of unique filenames to store	
SMB2_NUM_DIALECT	3	number of SMB2 dialects to store	
SMB1_NUM_DIALECT	3	number of SMB1 dialects to store	SMB1_DECODE=1
SMB1_DIAL_MAXLEN	32	maximum length for SMB1 dialects	SMB1_DECODE=1
SMB2_NUM_STAT	18	number of unique SMB2 header status to store	
SMB1_SAVE_DATA	0	Whether or not to save files	SMB1_DECODE=1
SMB2_SAVE_DATA	0	Whether or not to save files	
SMB_SAVE_AUTH	0	Whether or not to save NTLM authentications	
SMB_NATIVE_NAME_LEN	64	Maximum length for names	
SMB_SAVE_DIR	"/tmp/TranSMB/"	Folder for saved data	SMB_SAVE_DATA=1
SMB_AUTH_FILE	"smb_auth.txt"	File where to store NTLM authentications	SMB_SAVE_AUTH=1
SMB_RM_DATADIR	1	Whether to remove SMB_SAVE_DIR before starting	SMB_SAVE_DATA=1
SMB_FNAME_LEN	512	Maximum length for filenames	

When saving files, the plugin uses a combination of the file ID and the flow index as name. The file ID can be replaced with the real filename by using the smbrename script and the SMB_GUID_MAP_FILE (*smb_filenames.txt*) file (See Section 43.5).

43.3 Flow File Output

The smbDecode plugin outputs the following columns:

Column	Type	Description	Flags
smbStat	H16	Status	
smb1NDialects	U32	Number of requested dialects (SMB1)	
smb1Dialects	RS	SMB1 requested dialects (client: supported, server: chosen)	
smb2NDialects	U32	Number of dialects (SMB2)	
smb2Dialects	RH16	SMB2 dialect revision (client: supported, server: chosen)	
smbNHdrStat	U32	Number of unique SMB2 header status values	

Column	Type	Description	Flags
smbHdrStat	RH32	SMB2 list of uniq header status	
smbOpcodes	H32	Opcodes	
smbNOpcodes	19x(U32)	Number of records per opcode	
smbPrevSessId	H64	SMB previous session ID	
smbNativeOS	S	SMB native OS	
smbNativeLanMan	S	SMB native LAN Manager	
smbPrimDom	S	SMB primary domain	
smbTargName	S	SMB target name	
smbDomName	S	SMB domain name	
smbUserName	S	SMB user name	
smbHostName	S	SMB host name	
smbNTLMServChallenge	S	SMB NTLM server challenge	
smbNTProofStr	S	SMB NT proof string	
smbSessionKey	S	SMB session key	
smbGUID	S	Client/Server GUID	
smbSessFlags_	H16_	Session flags,	
secM_	H8_	Security mode,	
caps	H32	Capabilities	
smbBootT	TS	Server start time	
smbMaxSizeT_R_W	U32_U32_U32	Max transaction/read/write size	
smbPath	S	Full share path name	
smbShareT	H8	Type of share being accessed	
smbShareFlags	H32_	Share flags,	
caps	H32_	Capabilities,	
acc	H32	Access mask	
smbNFiles	U32	Number of accessed files	
smbFiles	RS	Accessed files	

43.3.1 smbStat

The smbStat column is to be interpreted as follows:

smbStat	Description
0x0001	Flow is SMB
0x0002	SMB2 header status list truncated... increase SMB2_NUM_STAT
0x0004	Dialect name truncated... increase SMB1_DIAL_MAXLEN
0x0008	SMB1 dialect list truncated... increase SMB1_NUM_DIALECT
0x0010	SMB2 dialect list truncated... increase SMB_NUM_DIALECT
0x0020	List of accessed files truncated... increase SMB_NUM_FNAME
0x0040	Selected dialect index out of bound... increase SMB1_NUM_DIALECT
0x0080	Selected dialect index out of bound (error or reverse flow not found)
0x0100	Filename truncated... increase SMB_FNAME_LEN
0x1000	Authentication information extracted
0x8000	Malformed packets

43.3.2 smb2Dialects

The `smb2Dialects` column is to be interpreted as follows:

smb2Dialects	Description
0x0202	SMB 2.0.2
0x0210	SMB 2.1
0x0300	SMB 3
0x0302	SMB 3.0.2
0x0311	SMB 3.1.1
0x02ff	Wildcard revision number ($\geq$ 2.1)

43.3.3 smbHdrStat

The `smbHdrStat` column is to be interpreted as follows:

smbOpcodes	Description
0x00000000	STATUS_SUCCESS
0x00000103	STATUS_PENDING
0x0000010b	STATUS_NOTIFY_CLEANUP
0x0000010c	STATUS_NOTIFY_ENUM_DIR
0x80000005	STATUS_BUFFER_OVERFLOW
0x80000006	STATUS_NO_MORE_FILES
0xc0000003	STATUS_INVALID_INFO_CLASS
0xc000000d	STATUS_INVALID_PARAMETER
0xc000000f	STATUS_NO_SUCH_FILE
0xc0000010	STATUS_INVALID_DEVICE_REQUEST
0xc0000011	STATUS_END_OF_FILE
0xc0000016	STATUS_MORE_PROCESSING_REQUIRED
0xc0000022	STATUS_ACCESS_DENIED
0xc0000023	STATUS_BUFFER_TOO_SMALL
0xc0000034	STATUS_OBJECT_NAME_NOT_FOUND
0xc0000035	STATUS_OBJECT_NAME_COLLISION
0xc000003a	STATUS_OBJECT_PATH_SYNTAX_BAD
0xc0000043	STATUS_SHARING_VIOLATION
0xc0000061	STATUS_PRIVILEGE_NOT_HELD
0xc000006a	STATUS_WRONG_PASSWORD
0xc000006d	STATUS_LOGON_FAILURE
0xc0000071	STATUS_PASSWORD_EXPIRED
0xc00000ac	STATUS_PIPE_NOT_AVAILABLE
0xc00000ba	STATUS_FILE_IS_A_DIRECTORY
0xc00000bb	STATUS_NOT_SUPPORTED
0xc00000c9	STATUS_NETWORK_NAME_DELETED
0xc00000cc	STATUS_BAD_NETWORK_NAME
0xc0000101	STATUS_DIRECTORY_NOT_EMPTY
0xc0000120	STATUS_CANCELLED
0xc0000128	STATUS_FILE_CLOSED

smbOpcodes	Description
0xc000019c	STATUS_FS_DRIVER_REQUIRED
0xc0000203	STATUS_USER_SESSION_DELETED
0xc0000225	STATUS_NOT_FOUND
0xc0000234	STATUS_ACCOUNT_LOCKED_OUT
0xc0000257	STATUS_PATH_NOT_COVERED
0xc0000275	STATUS_NOT_A_REPARSE_POINT

For a comprehensive list of the possible status and more extensive description, refer to [MS-ERREF], Section 2.3.1.

43.3.4 smbOpcodes

The `smbOpcodes` column is to be interpreted as follows:

smbOpcodes	Description
2^0 (=0x00000001)	SMB2_NEGOTIATE
2^1 (=0x00000002)	SMB2_SESSION_SETUP
2^2 (=0x00000004)	SMB2_LOGOFF
2^3 (=0x00000008)	SMB2_TREE_CONNECT
2^4 (=0x00000010)	SMB2_TREE_DISCONNECT
2^5 (=0x00000020)	SMB2_CREATE
2^6 (=0x00000040)	SMB2_CLOSE
2^7 (=0x00000080)	SMB2_FLUSH
2^8 (=0x00000100)	SMB2_READ
2^9 (=0x00000200)	SMB2_WRITE
2^{10} (=0x00000400)	SMB2_LOCK
2^{11} (=0x00000800)	SMB2_IOCTL
2^{12} (=0x00001000)	SMB2_CANCEL
2^{13} (=0x00002000)	SMB2_ECHO
2^{14} (=0x00004000)	SMB2_QUERY_DIRECTORY
2^{15} (=0x00008000)	SMB2_CHANGE_NOTIFY
2^{16} (=0x00010000)	SMB2_QUERY_INFO
2^{17} (=0x00020000)	SMB2_SET_INFO
2^{18} (=0x00040000)	SMB2_OPLOCK_BREAK

43.3.5 smbNOpcodes

The `smbNOpcodes` column reports the number of records of each type separated by underscores.

smbNOpcodes	Description
1	Number of SMB2_NEGOTIATE records
2	Number of SMB2_SESSION_SETUP records
3	Number of SMB2_LOGOFF records
4	Number of SMB2_TREE_CONNECT records
5	Number of SMB2_TREE_DISCONNECT records

smbNOpcodes	Description
6	Number of SMB2_CREATE records
7	Number of SMB2_CLOSE records
8	Number of SMB2_FLUSH records
9	Number of SMB2_READ records
10	Number of SMB2_WRITE records
11	Number of SMB2_LOCK records
12	Number of SMB2_IOCTL records
13	Number of SMB2_CANCEL records
14	Number of SMB2_ECHO records
15	Number of SMB2_QUERY_DIRECTORY records
16	Number of SMB2_CHANGE_NOTIFY records
17	Number of SMB2_QUERY_INFO records
18	Number of SMB2_SET_INFO records
19	Number of SMB2_OPLOCK_BREAK records

43.3.6 smbSessFlags_secM_caps

The `smbSessFlags_secM_caps` column is to be interpreted as follows:

smbSessFlags	Description
0x01	Client authenticated as guest user
0x02	Client authenticated as anonymous user
0x04	Server requires encryption of messages on this session (SMB 3.x)

smbSecM	Description
0x01	Security signatures enabled on the server
0x02	Security signatures required by the server

smbCaps	Description
0x01	Server supports the Distributed File System (DFS)
0x02	Server supports leasing
0x04	Server supports multi-credit operation (Large MTU)
0x08	Server supports establishing multiple channels for a single session
0x10	Server supports persistent handles
0x20	Server supports directory leasing
0x40	Server supports encryption

43.3.7 smbShareT

The `smbShareT` column is to be interpreted as follows:

smbShareT	Description
0x01	Physical disk share
0x02	Named pipe share
0x03	Printer share

43.3.8 smbShareFlags_caps_acc

The smbShareFlags_caps_acc column is to be interpreted as follows:

smbShareFlags	Description
0x00000001	Specified share is present in a Distributed File System (DFS) tree structure
0x00000002	Specified share is present in a DFS tree structure (DFS root)
	If none of the following three bits is set, then the caching policy is "manual"
0x00000010	Auto caching
0x00000020	VDO Caching
0x00000030	Offline caching MUST NOT occur
0x00000100	Restrict exclusive opens
0x00000200	Force shared delete
0x00000400	Allow namespace caching
0x00000800	Server will filter directory entries based on access permissions of the client
0x00001000	Server will not issue exclusive caching rights on this share
0x00002000	Enable hash V1
0x00004000	Enable hash V2
0x00008000	Encrypt data required

smbShareCaps	Description
0x00000008	Specified share is present in a DFS tree structure
0x00000010	Continuous availability
0x00000020	Scaleout
0x00000040	Cluster
0x00000080	Asymmetric

smbShareAcc	Description
0x00000001	Read access
0x00000002	Write access
0x00000004	Append access
0x00000008	Read extended attributes access
0x00000010	Write extended attributes access
0x00000020	Execute access
0x00000040	Delete child access

smbShareAcc	Description
0x00000080	Read attributes access
0x00000100	Write attributes access
0x00010000	Delete access
0x00020000	Read access to owner, group and ACL of the SID
0x00040000	Owner may write the DAC
0x00080000	Can write owner (take ownership)
0x00100000	Can wait on handle to synchronise on completion of I/O
0x01000000	System security is NOT set
0x02000000	Maximum allowed is NOT set
0x10000000	Generic all is NOT set
0x20000000	Generic execute is NOT set
0x40000000	Generic write is NOT set
0x80000000	Generic read is NOT set

43.4 Plugin Report Output

The number of SMB, SMB2 and SMB3 records is reported. In addition, if SMB_SAVE_AUTH=1, the number of NetNTLMv2 hashes extracted is reported.

43.5 Post-Processing

43.5.1 smbrename

The **smbrename** script can be used to rename and organise the files extracted by the plugin. It must be run from within the SMB_SAVE_DIR folder (where the file *smb_filenames.txt* is located). By default, it will replace the file ID with the real filename and organise the files into folders according to their mimetype. Either operation can be performed or not. Try 'smbrename -help' for more information.

43.5.2 SMB Authentications

When SMB1_DECODE=1, SMB_SECBLOB=1 and SMB_SAVE_AUTH=1, the plugin produces a file with suffix SMB_AUTH_FILE containing all the NetNTLMv2 hashes extracted from the traffic. The hashes can then be reversed using JohnTheRipper[6] or Hashcat[7] as follows:

```
john --wordlist=password.lst -format=netntlmv2 FILE_smb_auth.txt
        hashcat -m 5600 FILE_smb_auth.txt wordlist.txt
```

43.6 References

- [MS-CIFS]: Common Internet File System (CIFS) Protocol
- [MS-SMB]: Server Message Block (SMB) Protocol
- [MS-SMB2]: Server Message Block (SMB) Protocol Versions 2 and 3
- [MS-ERREF]: Windows Error Codes

[6] https://github.com/magnumripper/JohnTheRipper
[7] https://hashcat.net

- [MS-SPNG]: Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) Extension
- [MS-AUTHSOD]: Authentication Services Protocols Overview
- [MS-DTYP]: Windows Data Types
- [RFC4178]: The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism

44 smtpDecode

44.1 Description

The smtpDecode plugin processes MAIL header and content information of a flow. The idea is to identify certain mail features and CNAMES. User defined compiler switches are in *smtpDecode.h*.

44.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
SMTP_SAVE	0	1: save content to SMTP_F_PATH
SMTP_BTFLD	0	1: Bitfield coding of SMTP commands
SMTP_RCTXT	1	1: print response code text
SMTP_MXNMLN	70	maximal name length
SMTP_MXUNMLN	25	maximal user length
SMTP_MXPNMLN	15	maximal PW length
MAXCNM	8	maximal number rec,trans codes
MAXUNM	5	maxiaml number server names
MAXPNM	5	maxiaml number server names
MAXSNM	8	maximal number of server addresses
MAXRNM	8	maximal number of rec EMail addresses
MAXTNM	8	maximal number of trans EMail addresses

44.3 Flow File Output

The smtpDecode plugin outputs the following columns:

Column	Type	Description	Flags
smtpStat	H8	Status	
smtpCBF	H16	Command bit field	BITFIELD=1
smtpCC	RSC	Command Codes	
smtpRC	RI16	Response Codes	
smtpUsr	RS	SMTP Users	
smtpPW	RS	SMTP Passwords	
smtpSANum	I8	number of Server addresses	
smtpESANum	I8	number of email sender addresses	
smtpERANum	I8	number of email receiver addresses	
smtpSA	RS	Server send addresses	
smtpESA	RS	Email send addresses	
smtpERA	RS	Email receive addresses	

44.3.1 smtpStat

The smtpStat column describes the errors encountered during the flow lifetime:

smtpStat	Name	Description
2^0 (=0x01)	SMTP_INIT	SMTP ports found
2^1 (=0x02)	SMTP_AUTP	Authentication pending
2^2 (=0x04)	SMTP_DTP	data download pending, `SMTP_SAVE=1`
2^3 (=0x08)	PWSTATE	User PW pending
2^4 (=0x10)	SMTP_PWF	flow write finished, `SMTP_SAVE=1`
2^5 (=0x20)	—	—
2^6 (=0x40)	SMTP_FERR	File error, `SMTP_SAVE=1`
2^7 (=0x80)	SMTP_OVFL	array overflow

44.3.2 smtpCBF

The `smtpCBF` column is to be interpreted as follows:

smtpCBF	Description
2^0 (=`0x0001`)	HELO
2^1 (=`0x0002`)	EHLO
2^2 (=`0x0004`)	MAIL
2^3 (=`0x0008`)	RCPT
2^4 (=`0x0010`)	DATA
2^5 (=`0x0020`)	RSET
2^6 (=`0x0040`)	SEND
2^7 (=`0x0080`)	SOML
2^8 (=`0x0100`)	SAML
2^9 (=`0x0200`)	VRFY
2^{10} (=`0x0400`)	EXPN
2^{11} (=`0x0800`)	HELP
2^{12} (=`0x1000`)	NOOP
2^{13} (=`0x2000`)	QUIT
2^{14} (=`0x4000`)	TURN
2^{15} (=`0x8000`)	AUTH

44.4 TODO

- fragmentation

45 snmpDecode

45.1 Description

The snmpDecode plugin analyzes SNMP traffic.

45.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
SNMP_STRLEN	64	Maximum length for strings

45.3 Flow File Output

The snmpDecode plugin outputs the following columns:

Column	Type	Description
snmpStat	H8	Status
snmpVersion	U8	Version
snmpCommunity	S	Community (SNMPv1-2)
snmpUsername	S	Username (SNMPv3)
snmpMsgT	H16	Message types
snmpNumReq_Next_Resp_Set_Trap1_Bulk_Info_Trap2_Rep	U64_U64_U64_U64_U64_U64_U64_U64_U64	Number of GetRequest, GetNextRequest, GetResponse, SetRequest, Trapv1, GetBulkRequest, InformRequest, Trapv2, and Report packets

45.3.1 snmpStat

The snmpStat column is to be interpreted as follows:

snmpStat	Description
0x01	Flow is SNMP
0x40	String was truncated... increase SNMP_STRLEN
0x80	Packet was malformed

45.3.2 snmpVersion

The snmpVersion column is to be interpreted as follows:

snmpVersion	Description
0	SNMPv1
1	SNMPv2c
3	SNMPv3

45.3.3 snmpMsgT

The snmpMsgT column is to be interpreted as follows:

snmpMsgT	Description
0x0001	GetRequest
0x0002	GetNextRequest
0x0004	GetResponse
0x0008	SetRequest
0x0010	Trap (v1)
0x0020	GetBulkRequest (v2c, v3)
0x0040	InformRequest
0x0080	Trap (v2c, v3)
0x0100	Report

45.3.4 snmpType

The snmpType column is to be interpreted as follows:

snmpType	Description
0xa0	GetRequest
0xa1	GetNextRequest
0xa2	GetResponse
0xa3	SetRequest
0xa4	Trap (v1)
0xa5	GetBulkRequest (v2c, v3)
0xa6	InformRequest
0xa7	Trap (v2c, v3)
0xa8	Report

45.4 Packet File Output

In packet mode (-s option), the snmpDecode plugin outputs the following columns:

Column	Type	Description
snmpVersion	U8	Version
snmpCommunity	S	Community
snmpType	H8	Message type

45.5 Plugin Report Output

The following information is reported:

- Number of SNMP packets
- Number of SNMP GetRequest packets

- Number of SNMP GetNextRequest packets
- Number of SNMP GetResponse packets
- Number of SNMP SetRequest packets
- Number of SNMP Trap v1 packets
- Number of SNMP GetBulkRequest packets
- Number of SNMP InformRequest packets
- Number of SNMP Trap v2 packets
- Number of SNMP Report packets

46 socketSink

46.1 Description

This plugin is a socket interface of Tranalyzer. The idea is to interface one or many distributed Tranalyzer instances with a central server post-processing and visualising its data. The plugin also implements the Alarm Mode being activated by `ALARM_MODE=1` in the core *tranalyzer.h* file. Prepending information such as data length, checksum, or an id is controlled by the `BUF_DATA_SHFT` variable in the Tranalyzer core: *outputBuffer.h*. The user needs to configure the destination port, socket type and whether host info is transmitted in the first record. Otherwise the socketSink plugin requires no dependencies and produces output directly into the ETHERNET interface.

46.2 Dependencies

46.2.1 External Libraries

If gzip compression is activated (`GZ_COMPRESS=1`), then **zlib** must be installed.

Kali/Ubuntu: `sudo apt-get install zlib1g-dev`

Arch: `sudo pacman -S zlib`

Fedora/Red Hat: `sudo yum install zlib-devel`

Gentoo: `sudo emerge zlib`

OpenSUSE: `sudo zypper install zlib-devel`

Mac OS X: `brew install zlib`[8]

46.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description	Flags
`SERVADD`	127.0.0.1	destination address	
`DPORT`	6666	destination port (host order)	
`SOCKTYP`	1	Socket type: 0: UDP; 1: TCP	
`GZ_COMPRESS`	0	Whether or not to compress the output (gzip)	`SOCKTYP=1`
`CONTENT_TYPE`	1	0: binary; 1: text; 2: json	
`HOST_INFO`	0	0: no info; 1: all info about host	`CONTENT_TYPE=1`

46.3.1 bin2txt.h

`bin2txt.h` controls the conversion from internal binary format to standard text output.

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

Variable	Default	Description
HEX_CAPITAL	0	Hex number representation: 0: lower case, 1: upper case
IP4_NORMALIZE	0	IPv4 addresses representation: 0: normal, 1: normalized (padded with 0)
IP6_COMPRESS	1	IPv6 addresses representation: 1: compressed, 0: full 128 bit length
TFS_EXTENDED_HEADER	0	Whether or not to print an extended header in the flow file (number of rows, columns, columns type)
B2T_LOCALTIME	0	Time representation: 0: UTC, 1: localtime
B2T_TIME_IN_MICRO_SECS	1	Time precision: 0: nanosecs, 1: microsecs
HDR_CHR	"%"	start character of comments in flow file
SEP_CHR	"\t"	character to use to separate the columns in the flow file

46.4 Additional Output

The output buffer normally being written to the flow file will be directed to the socket.

If HOST_INFO=1 then the following header is transmitted as a prelude.

Parameter	Type	Description
1	U32	Message length, if BUF_DATA_SHFT > 0
2	U32	Checksum, if BUF_DATA_SHFT > 1
3	U32	Sensor ID
4	U64.U32	Present Unix timestamp
5	RS;	OS;Machine Name;built;OS type;HW;
	RS;	Ethername1(address1)Ethername2(address2)...;
	RS;	IPInterfacename1(address1/netmask1)IPInterfacename2(address2/netmask2)...;

After the prelude all flow based binary buffer will be directed to the socket interface according to the format shown in the following table:

Column	Type	Description
1	U32	Message length, if BUF_DATA_SHFT > 0
2	U32	Checksum, if BUF_DATA_SHFT > 1
3	RU32	Binary buffer output

46.5 Example

1. Open a socket, e.g., with netcat: `nc -l 127.0.0.1 6666`
2. Start T2 with the socketSink plugin, e.g., `t2 -r file.pcap`
3. You should now see the flows on your netcat terminal

To simulate a server collecting data from many T2 or save the transmitted flows into a file, use the following command: `nc -l 127.0.0.1 6666 > flowfile.txt`

47 sqliteSink

47.1 Description

The sqliteSink plugin outputs flow files to SQLite database.

47.2 Dependencies

47.2.1 External Libraries

This plugin depends on the **sqlite** library.

Ubuntu: `sudo apt-get install libsqlite3-dev`

Arch: `sudo pacman -S sqlite`

Mac OS X: `brew install sqlite`

47.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
SQLITE_OVERWRITE	2	0: abort if table already exists 1: overwrite table if it already exists 2: append to table if it already exists
SQLITE_HEX_AS_INT	0	0: store hex numbers (bitfields) as text 1: store hex numbers (bitfields) as int
SQLITE_TRANSACTION_NFLOWS	40000	0: one transaction > 0: one transaction every *n* flows
SQLITE_QRY_LEN	32768	Max length for query
SQLITE_DBNAME	"/tmp/tranalyzer.db"	Name of the database
SQLITE_TABLE_NAME	"flow"	Name of the table

48 sshDecode

48.1 Description

This plugin analyzes SSH traffic.

48.2 Dependencies

This plugin requires the **libssl**.

Arch: `sudo pacman -S openssl`

Ubuntu/Kali: `sudo apt-get install libssl-dev`

OpenSUSE: `sudo zypper install libopenssl-devel`

Red Hat/Fedora: `sudo yum install openssl-devel`

Mac OSX: `brew install openssl`

48.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
SSH_USE_PORT	1	Whether (1) or not (0) to count all packets to/from SSH_PORT as SSH (useful if version exchange was not captured)
SSH_DECODE	0	Decode SSH handshake messages (experimental)
SSH_DEBUG	0	Activate debug output

48.4 Flow File Output

The sshDecode plugin outputs the following columns:

Column	Type	Description
sshStat	H8	Status
sshVersion	RS	SSH version and software
If SSH_DECODE=1, the following columns are displayed:		
sshFingerprint	RS	SSH public key fingerprint
sshCookie	RS	SSH cookie
sshKEX	RS	SSH KEX Algorithms
sshSrvHostKeyAlgo	RS	SSH server host key algorithms
sshEncCS	RS	SSH encryption algorithms client to server
sshEncSC	RS	SSH encryption algorithms server to client

Column	Type	Description
sshMacCS	RS	SSH MAC algorithms client to server
sshMacSC	RS	SSH MAC algorithms server to client
sshCompCS	RS	SSH compression algorithms client to server
sshCompSC	RS	SSH compression algorithms server to client
sshLangCS	RS	SSH languages client to server
sshLangSC	RS	SSH languages server to client

48.4.1 sshStat

The sshStat column is to be interpreted as follows:

sshStat	Description
0x01	Flow contains SSH protocol
0x02	Keeps track of who sent the SSH banner first
0x40	SSH version got truncated
0x80	Banner does not end with CRLF or contains NULL byte

48.5 Plugin Report Output

The number of SSH flows is reported.

49 sslDecode

49.1 Description

This plugin analyzes SSL/TLS and OpenVPN traffic.

49.2 Dependencies

If `SSL_ANALYZE_CERT` is activated, then **libssl** is required.

Arch: `sudo pacman -S openssl`

Ubuntu/Kali: `sudo apt-get install libssl-dev`

OpenSUSE: `sudo zypper install libopenssl-devel`

Red Hat/Fedora: `sudo yum install openssl-devel`

Mac OSX: `brew install openssl`

49.3 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
SSL_ANALYZE_OVPN	0	Analyze OpenVPN (Experimental)
SSL_EXT_LIST	1	Output the list and number of extensions
SSL_MAX_EXT	8	Maximum number of extensions to store
SSL_EC	1	Output the list and number of elliptic curves
SSL_MAX_EC	6	Maximum number of elliptic curves to store
SSL_EC_FORMATS	1	Output the list and number of elliptic curve formats
SSL_MAX_EC_FORMATS	6	Maximum number of elliptic curve formats to store
SSL_PROTO_LIST	1	Output the list and number of protocols
SSL_MAX_PROTO	6	Maximum number of protocols to store
SSL_PROTO_LEN	16	Maximum number of characters per protocol
SSL_CIPHER_LIST	1	Output the list and number of supported ciphers
SSL_MAX_CIPHER	3	Maximum number of ciphers to store
SSL_ANALYZE_CERT	1	Analyze certificates

If `SSL_ANALYZE_CERT > 0`, the following flags are available:

Name	Default	Description
SSL_CERT_SERIAL	1	Print the certificate serial number
SSL_CERT_FINGPRINT	1	0: no certificate fingerprint, 1: SHA1, 2: MD5
SSL_CERT_VALIDITY	1	Print certificates validity (Valid from/to, lifetime)
SSL_CERT_SIG_ALG	1	Print the certificate signature algorithm
SSL_CERT_PUBKEY_ALG	1	Print the certificate public key algorithm
SSL_CERT_ALG_NAME_LONG	0	Whether to use short (0) or long (1) names for algorithms
SSL_CERT_PUBKEY_TS	1	Print certificates public key type and size
SSL_CERT_SUBJECT	2	0: no info about cert subject, 1: whole subject as one string, 2: selected fields (see below)
SSL_CERT_ISSUER	2	0: no info about cert issuer, 1: whole issuer as one string, 2: selected fields (see below)
SSL_CERT_COMMON_NAME	1	Print the common name of the issuer/subject
SSL_CERT_ORGANIZATION	1	Print the organization name of the issuer/subject
SSL_CERT_ORG_UNIT	1	Print the organizational unit of the issuer/subject
SSL_CERT_LOCALITY	1	Print the locality name of the issuer/subject
SSL_CERT_STATE	1	Print the state/province name of the issuer/subject
SSL_CERT_COUNTRY	1	Print the country of the issuer/subject (iso3166)
SSL_RM_CERTDIR	1	Remove SSL_CERT_PATH before starting
SSL_SAVE_CERT	0	Save certificates
SSL_CERT_NAME_FINDEX	0	Prepend the flowIndex to the certificate name
SSL_BLIST	0	Flag blacklisted certificates
SSL_JA3	1	Output JA3 fingerprints (hash and description)
SSL_JA3_STR	0	Also output JA3 fingerprints before hashing

If SSL_SAVE_CERT==1 then, certificates are saved under SSL_CERT_PATH (default: /tmp/TranCerts/) with the extension SSL_CERT_EXT (default: .pem) and the SHA1 or MD5 fingerprint as filename.

49.4 Flow File Output

The sslDecode plugin outputs the following columns:

Column	Type	Description	Flags
sslStat	H16	Status	
sslProto	H16	Protocol	
ovpnType	H16	OpenVPN message types	SSL_ANALYZE_OVPN=1
ovpnSessionID	U64	OpenVPN session ID	SSL_ANALYZE_OVPN=1

Column	Type	Description	Flags
sslFlags	H8	SSL flags	
sslVersion	H16	SSL/TLS Version	
sslVuln	H8	Vulnerabilities	
sslAlert	H32	Alert type	
sslCipher	H16	Preferred (Client)/Negotiated (Server) cipher	
sslNumExt	U16	Number of extensions	SSL_EXT_LIST=1
sslExtList	RH16	List of extensions	SSL_EXT_LIST=1
sslNumECPt	U16	Number of elliptic curve points	SSL_EC=1
sslECPt	RH16	List of elliptic curve points	SSL_EC=1
sslNumECFormats	U8	Number of EC point formats	SSL_EC_FORMATS=1
sslECFormats	RH8	List of EC point formats	SSL_EC_FORMATS=1
sslNumProto	U16	Number of protocols	SSL_PROTO_LIST=1
sslProtoList	RS	List of protocols	SSL_PROTO_LIST=1
sslNumCipher	U16	Number of supported ciphers	SSL_CIPHER_LIST=1
sslCipherList	RH16	List of supported ciphers	SSL_CIPHER_LIST=1
sslNumCC_	U16_	Number of change_cipher records,	
A_	U16_	Number of alert records,	
H_	U16_	Number of handshake records,	
AD_	U64_	Number of application data records,	
HB	U64	Number of heartbeat records	
sslSessIdLen	U8	Session ID length	
sslGMTTime	RTS	GMT Unix Time	
sslServerName	RS	server name	

If SSL_ANALYZE_CERT == 1, the following columns are output:

sslCertVersion	RU8	Certificate version	SSL_CERT_FINGPRINT=1
sslCertSerial	RSC	Certificate serial number	SSL_CERT_SERIAL=1
sslCertSha1FP	RSC	Certificate SHA1 fingerprint	SSL_CERT_FINGPRINT=1
sslCertMd5FP	RSC	Certificate MD5 fingerprint	SSL_CERT_FINGPRINT=2
sslCNotValidBefore_	TS_	Certificate validity: not valid before,	SSL_CERT_VALIDITY=1
after_	TS_	not valid after,	
lifetime	U64	lifetime	
sslCSigAlg	RS	Certificate signature algorithm	SSL_CERT_SIG_ALG=1
sslCKeyAlg	RS	Certificate public key algorithm	SSL_CERT_PUBKEY_ALG=1
sslCPKeyType_	SC_	Certificate public key type,	SSL_CERT_PUBKEY_TS=1
Size	U16	Certificate public key size (bits)	

If SSL_CERT_SUBJECT > 0, the following columns are output:

sslCSubject	RS	Certificate subject	SSL_CERT_SUBJECT=1
sslCSubjectCommonName	RS	Certificate subject common name	SSL_CERT_SUBJECT=2
sslCSubjectOrgName	RS	Certificate subject organization name	SSL_CERT_SUBJECT=2
sslCSubjectOrgUnit	RS	Certificate subject organizational unit name	SSL_CERT_SUBJECT=2
sslCSubjectLocality	RS	Certificate subject locality name	SSL_CERT_SUBJECT=2

Column	Type	Description	Flags
sslCSubjectState	RS	Certificate subject state or province name	SSL_CERT_SUBJECT=2
sslCSubjectCountry	RS	Certificate subject country name	SSL_CERT_SUBJECT=2
If SSL_CERT_ISSUER > 0, the following columns are output:			
sslCIssuer	RS	Certificate issuer	SSL_CERT_ISSUER=1
sslCIssuerCommonName	RS	Certificate issuer common name	SSL_CERT_ISSUER=2
sslCIssuerOrgName	RS	Certificate issuer organization name	SSL_CERT_ISSUER=2
sslCIssuerOrgUnit	RS	Certificate issuer organizational unit name	SSL_CERT_ISSUER=2
sslCIssuerLocality	RS	Certificate issuer locality name	SSL_CERT_ISSUER=2
sslCIssuerState	RS	Certificate issuer state or province name	SSL_CERT_ISSUER=2
sslCIssuerCountry	RS	Certificate issuer country name	SSL_CERT_ISSUER=2
sslBlistCat	RS	Blacklisted certificate category	SSL_BLIST=1
sslJA3Hash	RSC	JA3 fingerprint	SSL_JA3=1
sslJA3Desc	RS	JA3 description	SSL_JA3=1
sslJA3Str	RS	JA3 string	SSL_JA3=1&& SSL_JA3_STR=1

If SSL_CERT_SUBJECT=2 or SSL_CERT_ISSUER=2, then the columns displayed are controlled by the following self-explanatory flags:

- SSL_CERT_COMMON_NAME,
- SSL_CERT_ORGANIZATION,
- SSL_CERT_ORG_UNIT,
- SSL_CERT_LOCALITY,
- SSL_CERT_STATE,
- SSL_CERT_COUNTRY.

49.4.1 sslStat

The hex based status variable `sslStat` is defined as follows:

sslStat	Description
0x0001	message had mismatched version
0x0002	record was too long (max 16384)
0x0004	record was malformed, eg, invalid value
0x0008	certificate had expired
0x0010	connection was closed due to fatal alert
0x0020	connection was renegotiated (existed before)
0x0040	peer not allowed to send heartbeat requests
0x0080	cipher list truncated... increase SSL_MAX_CIPHER
0x0100	extension list truncated... increase SSL_MAX_EXT
0x0200	protocol list truncated... increase SSL_MAX_PROTO
0x0400	protocol name truncated... increase SSL_PROTO_LEN
0x0800	EC or EC formats list truncated... increase SSL_MAX_EC or SSL_MAX_EC_FORMATS
0x1000	Certificate is blacklisted
0x2000	weak cipher detected (Null, DES, RC4 (RFC7465), ADH, 40/56 bits)
0x4000	weak protocol detected (SSL 2.0, SSL 3.0)
0x8000	weak key detected

49.4.2 sslProto

The hex based protocol variable `sslProto` is defined as follows:

sslProto	Description
0x0001	HTTP/0.9, HTTP/1.0, HTTP/1.1 (ALPN starts with http)
0x0002	HTTP/2 (h2, h2c)
0x0004	HTTP/3 (h3)
0x0008	SPDY
0x0010	IMAP
0x0020	POP3
0x0040	FTP
0x0080	XMPP jabber
0x0100	STUN/TURN
0x0200	APNS (Apple Push Notification Service)
0x0400	WebRTC Media and Data
0x0800	CoAP
0x1000	ManageSieve
0x2000	RTP or RTCP[9]
0x4000	OpenVPN[10]

[9] Guessed by the presence of the use-srtp hello extension
[10] Guessed by being able to decode the protocol

sslProto	Description
0x8000	Unknown protocol (ALPN matched none of the above)

49.4.3 ovpnType

The ovpnType column is to be interpreted as follows:

ovpnType	Description
2^1 (=0x0002)	P_CONTROL_HARD_RESET_CLIENT_V1
2^2 (=0x0004)	P_CONTROL_HARD_RESET_SERVER_V1
2^3 (=0x0008)	P_CONTROL_SOFT_RESET_V1
2^4 (=0x0010)	P_CONTROL_V1
2^5 (=0x0020)	P_ACK_V1
2^6 (=0x0040)	P_DATA_V1
2^7 (=0x0080)	P_CONTROL_HARD_RESET_CLIENT_V2
2^8 (=0x0100)	P_CONTROL_HARD_RESET_SERVER_V2
2^9 (=0x0200)	P_DATA_V2

49.4.4 sslFlags

The sslFlags is defined as follows:

sslFlags	Description
0x01	request is SSLv2
0x02	SSLv3 version on 'request' layer different than on 'record' layer
0x04	gmt_unix_time is small (less than 1 year since epoch, probably seconds since boot)
0x08	gmt_unix_time is more than 5 years in the future (probably random)
0x10	random data (28 bytes) is not random
0x20	compression (deflate) is enabled

49.4.5 sslVersion

The hex based version variable sslVersion is defined as follows:

sslVersion	Description
0x0300	SSL 3.0
0x0301	TLS 1.0
0x0302	TLS 1.1
0x0303	TLS 1.2
0x0304	TLS 1.3
0xfefd	DTLS 1.2
0xfeff	DTLS 1.0

49.4.6 sslVuln

The hex based vulnerability variable `sslVuln` is defined as follows:

sslVuln	Description
0x01	vulnerable to BEAST
0x02	vulnerable to BREACH
0x04	vulnerable to CRIME
0x08	vulnerable to FREAK
0x10	vulnerable to POODLE
0x20	HEARTBLEED attack attempted
0x40	HEARTBLEED attack successful (Not implemented)

49.4.7 sslAlert

The hex based alert variable `sslAlert` is defined as follows:

sslAlert	Description
0x00000001	close notify
0x00000002	unexpected message
0x00000004	bad record MAC
0x00000008	decryption failed
0x00000010	record overflow
0x00000020	decompression failed
0x00000040	handshake failed
0x00000080	no certificate
0x00000100	bad certificate
0x00000200	unsupported certificate
0x00000400	certificate revoked
0x00000800	certificate expired
0x00001000	certificate unknown
0x00002000	illegal parameter
0x00004000	unknown CA
0x00008000	access denied

sslAlert	Description
0x00010000	decode error
0x00020000	decrypt error
0x00040000	export restriction
0x00080000	protocol version
0x00100000	insufficient security
0x00200000	internal error
0x00400000	user canceled
0x00800000	no renegotiation
0x01000000	unsupported extension
0x02000000	inappropriate fallback
0x04000000	certificate unobtainable
0x08000000	unrecognized name
0x10000000	bad certificate status response
0x20000000	bad certificate hash value
0x40000000	unknown PSK identity
0x80000000	no application protocol

49.4.8 sslCipher

The `sslCipher` variable represents the preferred cipher for the client and the negotiated cipher for the server. The corresponding name can be found in the *src/sslCipher.h* file.

49.4.9 sslNumCC_A_H_AD_HB

The number of message variable `sslNumCC_A_H_AD_HB` decomposed as follows:

sslNumCC_A_H_AD_HB	Description
sslNumCC	number of change cipher records

sslNumCC_A_H_AD_HB	Description
sslNumA	number of alerts records
sslNumH	number of handshake records
sslNumAD	number of application data records
sslNumHB	number of heartbeat records

49.4.10 sslExtList

The list of extensions is to be interpreted as follows:

sslExt	Description
0x0000	Server name
0x0001	Max fragment length
0x0002	Client certificate URL
0x0003	Trusted CA keys
0x0004	Truncated HMAC
0x0005	Status request
0x0006	User mapping
0x0007	Client authz
0x0008	Server authz
0x0009	Cert type
0x000a	Supported groups (elliptic curves)
0x000b	EC point formats
0x000c	SRP
0x000d	Signature algorithms
0x000e	Use SRTP
0x000f	Heartbeat

sslExt	Description
0x0010	ALPN
0x0011	Status request v2
0x0012	Signed certificate timestamp
0x0013	Client certificate type
0x0014	Server certificate type
0x0015	Padding
0x0016	Encrypt then MAC
0x0017	Extended master secret type
0x0023	Session ticket
0x0028	Extended random
0x3374	NPN
0x3377	Origin bound cert
0x337c	Encrypted client cert
0x754f	Channel ID old
0x7550	Channel ID
0xff01	renegotiation_info

49.4.11 sslCNotValidBefore_after_lifetime

The sslCNotValidBefore_after_lifetime indicates the validity period of the certificate, i.e., not valid before / after, and the number of seconds between those two dates.

49.5 Plugin Report Output

The number of OpenVPN, Tor, SSL 2.0, 3.0, TLS 1.0, 1.1, 1.2 and 1.3 and DTLS 1.0 (OpenSSL pre 0.9.8f), 1.0 and 1.2 flows is reported.

49.6 TODO

In order to analyze all certificates, we need to reassemble packets.

50 stpDecode

50.1 Description

The stpDecode plugin analyzes STP traffic.

50.2 Flow File Output

The stpDecode plugin outputs the following columns:

Column	Type	Description
stpStat	H8	Status
stpVersion	U8	Protocol Version Identifier
stpType	H8	Aggregated BPDU Types
stpFlags	H8	Aggregated BPDU flags

50.2.1 stpStat

The stpStat column is to be interpreted as follows:

stpStat	Description
0x01	Flow is STP

50.2.2 stpProto

The stpProto column is to be interpreted as follows:

stpProto	Description
0x0000	Spanning Tree Protocol

50.2.3 stpVersion

The stpVersion column is to be interpreted as follows:

stpVersion	Description
0	Spanning Tree
2	Rapid Spanning Tree
3	Multiple Spanning Tree
4	Shortest Path Tree

50.2.4 stpType

The stpType column is to be interpreted as follows:

stpType	Description
0x00	Configuration
0x02	Rapid/Multiple Spanning Tree
0x80	Topology Change Notification

50.2.5 stpFlags

The stpFlags column is to be interpreted as follows:

stpFlags	Description
2^0 (=0x01)	Topology Change Acknowledgment
2^1 (=0x02)	Agreement
2^2 (=0x04)	Forwarding
2^3 (=0x08)	Learning
2^4 (=0x10) 2^5 (=0x20)	Port Role: 0x00: Unknown, 0x10: Alternate or Backup, 0x20: Root, 0x30: Designated
2^6 (=0x40)	Proposal
2^7 (=0x80)	Topology Change

50.3 Packet File Output

In packet mode (-s option), the stpDecode plugin outputs the following columns:

Column	Type	Description
stpProto	H16	Protocol Identifier
stpVersion	U8	Protocol Version Identifier
stpType	H8	BPDU Type
stpFlags	H8	BPDU flags
stpRootPrio	U16	Root Priority
stpRootHw	MAC	Root System ID
stpRootCost	U32	Root Path Cost
stpBridgePrio	U16	Bridge Priority
stpBridgeHw	MAC	Bridge System ID
stpPort	H16	Port Identifier
stpMsgAge	U16	Message Age
stpMaxAge	U16	Max Age
stpHello	U16	Hello Time
stpForward	U16	Forward Delay
stpPvstOrigVlan	U16	Originating VLAN (PVSTP+)

50.4 Plugin Report Output

The number of STP packets is reported.

51 stunDecode

This plugin analyzes STUN, TURN and NAT-PMP traffic.

51.1 Required Files

None

51.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
NAT_PMP	1	Whether (1) or not (0) to analyse NAT-PMP

51.3 Flow File Output

The stunDecode plugin outputs the following columns:

Column	Type	Description
natStat	H32	status
natErr	H32	error code
natMCReq_Ind_Succ_Err	U16_U16_U16_U16	number of messages (Req, Ind, Succ, Err)
natAddr_Port	IP4_U16	mapped address and port
natXAddr_Port	IP4_U16	(xor) mapped address and port
natPeerAddr_Port	IP4_U16	peer address and port
natOrigAddr_Port	IP4_U16	response origin address and port
natRelayAddr_Port	IP4_U16	relayed address and port
natDstAddr_Port	IP4_U16	destination address and port
natOtherAddr_Port	IP4_U16	other address and port
natLifetime	U32	binding lifetime (seconds)
natUser	S	username
natPass	S	password
natRealm	S	realm
natSoftware	S	software
If NAT_PMP=1, the following columns are displayed:		
natPMPReqEA_MU_MT	U16_U16_U16	NAT-PMP num. of requests (External Address, Map UDP, Map TCP)
natPMPRespEA_MU_MT	U16_U16_U16	NAT-PMP num. of responses (External Address, Map UDP, Map TCP)
natPMPSSSOE	U32	NAT-PMP seconds since start of epoch

51.3.1 natStat

The `natStat` column is to be interpreted as follows:

natStat	Description
2^0 (=0x0000 0001)	STUN protocol
2^1 (=0x0000 0002)	TURN protocol
2^2 (=0x0000 0004)	ICE protocol
2^3 (=0x0000 0008)	SIP protocol
2^4 (=0x0000 0010)	Microsoft Extension
2^5 (=0x0000 0020)	Even Port
2^6 (=0x0000 0040)	Reserve next port
2^7 (=0x0000 0080)	don't fragment
2^8 (=0x0000 0100)	nonce
2^{13} (=0x0000 2000)	deprecated message attribute
2^{14} (=0x0000 4000)	STUN over non-standard port
2^{15} (=0x0000 8000)	malformed message
2^{16} (=0x0001 0000)	Port Mapping Protocol (PMP)
2^{31} (=0x8000 0000)	Packet snapped, analysis incomplete

51.3.2 natErr

The hex based error variable `natErr` is defined as follows (STUN):

natErr	Description
2^0 (=0x00000001)	try alt
2^1 (=0x00000002)	bad request
2^2 (=0x00000004)	unauthorized
2^3 (=0x00000008)	forbidden
2^4 (=0x00000010)	unknown attribute
2^5 (=0x00000020)	allocation mismatch
2^5 (=0x00000040)	stale nonce
2^6 (=0x00000080)	address family not supported
2^7 (=0x00000100)	wrong credentials
2^8 (=0x00000200)	unsupported transport protocol
2^9 (=0x00000400)	peer address family mismatch
2^{10} (=0x00000800)	connection already exists
2^{11} (=0x00001000)	connection timeout or failure
2^{12} (=0x00002000)	allocation quota reached
2^{13} (=0x00004000)	role conflict
2^{14} (=0x00008000)	server error
2^{15} (=0x00010000)	insufficient capacity
2^{31} (=0x80000000)	Unhandled error

The hex based error variable `natErr` is defined as follows (NAT-PMP):

natErr	Description
2^1 (=0x00000002)	Unsupported version
2^2 (=0x00000004)	Not authorized/refused
2^3 (=0x00000008)	Network failure
2^4 (=0x00000010)	Out of resources
2^5 (=0x00000020)	Unsupported opcode

51.3.3 natMCReq_Ind_Succ_Err

The number of messages variable `natMCReq_Ind_Succ_Err` decomposed as follows:

natMCReq_Ind_Succ_Err	Description
natMCReq	number of requests
natMCInd	number of indications
natMCSucc	number of success response
natMCErr	number of error response

51.4 TODO

Port Control Protocol (PCP)

52 syslogDecode

52.1 Description

The syslogDecode plugin analyzes Syslog traffic.

52.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
No configuration options available		

52.3 Flow File Output

The syslogDecode plugin outputs the following columns:

Column	Type	Description
syslogStat	H8	Status
syslogMCnt	U32	message count
syslogSev_Fac_Cnt	RU8_U8_U16	Number of severity/facility messages

52.3.1 syslogStat

The syslogStat column is to be interpreted as follows:

syslogStat	Description
0x01	Syslog detected
0x80	Counter for facility/severity overflowed

52.4 TODO

- IPv6 tests

52.5 References

- https://tools.ietf.org/html/rfc5424

53 tcpFlags

53.1 Description

The tcpFlags plugin contains IP and TCP header information encountered during the lifetime of a flow.

53.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
SPKTMD_SEQACKREL	0	Seq/Ack Numbers 0: absolute, 1: relative (-s option)
RTT_ESTIMATE	1	Whether (1) or not (0) to estimate Round trip time
IPCHECKSUM	2	0: No checksums calculation 1: Calculation of L3 (IP) Header Checksum 2: L3/L4 (TCP, UDP, ICMP, IGMP, ...) Checksum
WINDOWSIZE	1	Whether (1) or not (0) to output TCP window size parameters
SEQ_ACK_NUM	1	Whether (1) or not (0) to output Sequence/Acknowledge Number features
FRAG_ANALYZE	1	Whether (1) or not (0) to enable fragmentation analysis
NAT_BT_EST	1	Whether (1) or not (0) to estimate NAT boot time
SCAN_DETECTOR	1	Whether (1) or not (0) to enable scan flow detector
WINMIN	1	Minimal window size defining a healthy communication, below packets are counted

53.2.1 WINMIN

WINMIN default 1 setting selects all packets/flow where communication came to a halt due to receiver buffer overflow. Literally the number of window size 0 packets to the sender are then counted. WINMIN can be set to any value defining a healthy communication, which depends on the network and application.

53.3 Flow File Output

The tcpFlags plugin outputs the following columns:

Column	Type	Description	Flags
tcpFStat	H16	Status	
ipMindIPID	U16	IP minimum delta IP ID	
ipMaxdIPID	U16	IP maximum delta IP ID	
ipMinTTL	U8	IP minimum TTL	
ipMaxTTL	U8	IP maximum TTL	
ipTTLChg	U8	IP TTL Change Count	
ipTOS	H8	IP Type of Service	
ipFlags	H16	IP aggregated flags	
ipOptCnt	U16	IP options count	IPV6_ACTIVATE=0
ipOptCpCl_Num	H8_H32	IP aggregated options, copy-class and number	IPV6_ACTIVATE=0
ip6OptCntHH_D	U16_U16	IPv6 aggregated hop by hop dest. option counts	IPV6_ACTIVATE=1

Column	Type	Description	Flags
ip6OptHH_D	H32_H32	IPv6 hop by hop destination options	IPV6_ACTIVATE=1
tcpPSeqCnt	U16	TCP packet sequence count	SEQ_ACK_NUM=1
tcpSeqSntBytes	U64	TCP sent seq diff bytes	SEQ_ACK_NUM=1
tcpSeqFaultCnt	U16	TCP sequence number fault count	SEQ_ACK_NUM=1
tcpPAckCnt	U16	TCP packet ack count	SEQ_ACK_NUM=1
tcpFlwLssAckRcvdBytes	U64	TCP flawless ack received bytes	SEQ_ACK_NUM=1
tcpAckFaultCnt	U16	TCP ack number fault count	SEQ_ACK_NUM=1
tcpInitWinSz	U32	TCP initial effective window size	WINDOWSIZE=1
tcpAveWinSz	F	TCP average effective window size	WINDOWSIZE=1
tcpMinWinSz	U32	TCP minimum effective window size	WINDOWSIZE=1
tcpMaxWinSz	U32	TCP maximum effective window size	WINDOWSIZE=1
tcpWinSzDwnCnt	U16	TCP effective window size change down count	WINDOWSIZE=1
tcpWinSzUpCnt	U16	TCP effective window size change up count	WINDOWSIZE=1
tcpWinSzChgDirCnt	U16	TCP effective window size direction change count	WINDOWSIZE=1
tcpWinSzThRt	F	TCP packet count ratio below window size WINMIN	WINDOWSIZE=1
tcpFlags	H8	TCP aggregated protocol flags (CWR, ACK, PSH, RST, SYN, FIN)	
tcpAnomaly	H16	TCP aggregated header anomaly flags	
tcpOptPktCnt	U16	TCP options packet count	
tcpOptCnt	U16	TCP options count	
tcpOptions	H32	TCP aggregated options	
tcpMSS	U16	TCP Maximum Segment Length	
tcpWS	U8	TCP Window Scale	
tcpTmS	U32	TCP Time Stamp	NAT_BT_EST=1
tcpTmER	U32	TCP Time Echo Reply	NAT_BT_EST=1
tcpEcI	F	TCP Estimated counter increment	NAT_BT_EST=1
tcpBtm	TS	TCP Estimated Boot time	NAT_BT_EST=1
tcpSSASAATrip	F	(A) TCP Trip Time SYN, SYN-ACK, (B) TCP Trip Time SYN-ACK, ACK	RTT_ESTIMATE=1
tcpRTTAckTripMin	F	TCP Ack Trip Minimum	RTT_ESTIMATE=1
tcpRTTAckTripMax	F	TCP Ack Trip Maximum	RTT_ESTIMATE=1
tcpRTTAckTripAve	F	TCP Ack Trip Average	RTT_ESTIMATE=1
tcpRTTAckTripJitAve	F	TCP Ack Trip Jitter Average	RTT_ESTIMATE=1
tcpRTTSseqAA	F	(A) TCP Round Trip Time SYN, SYN-ACK, ACK (B) TCP Round Trip Time ACK-ACK RTT	RTT_ESTIMATE=1 RTT_ESTIMATE=1
tcpRTTAckJitAve	F	TCP Ack Round trip average Jitter	RTT_ESTIMATE=1

53.3.1 tcpFStat

The tcpFStat column is to be interpreted as follows:

tcpFStat	Description
0x0001	Packet no good for interdistance assessment
0x0002	Scan detected in flow
0x0004	Successful scan detected in flow
0x0008	Timestamp option decreasing
0x0010	TCP option init
0x0020	ACK packet loss state machine init
0x0040	Window state machine initialized
0x0080	Window state machine count up/down
0x0100	L4 checksum calculation if present
0x0200	UDP-Lite checksum coverage error

53.3.2 ipFlags

The ipFlags column is to be interpreted as follows:

ipFlags	Description
0x0001	IP options corrupt
0x0002	IPv4 packets out of order
0x0004	IPv4 ID roll over
0x0008	IP fragment below minimum
0x0010	IP fragment out of range
0x0020	More Fragment bit
0x0040	IPv4: Dont Fragment bit IPv6: reserve bit
0x0080	Reserve bit

ipFlags	Description
0x0100	Fragmentation position error
0x0200	Fragmentation sequence error
0x0400	L3 checksum error
0x0800	L4 checksum error
0x1000	L3 header length snapped
0x2000	Packet interdistance = 0
0x4000	Packet interdistance < 0
0x8000	TCP SYN flag with L7 content

53.3.3 ipOptCpCl_Num

The aggregated IP options are coded as a bit field in hexadecimal notation where the bit position denotes the IP options type according to following format: [$2^{\text{Copy-Class}}$]_[2^{Number}]. If the field reads: 0x10_0x00100000 in an ICMP message it is a 0x94 = 148 router alert.
Refer to RFC for decoding the bitfield: http://www.iana.org/assignments/ip-parameters.

53.3.4 tcpFlags

The tcpFlags column is to be interpreted as follows:

tcpFlags	Flag	Description
2^0 (=0x01)	FIN	No more data, finish connection
2^1 (=0x02)	SYN	Synchronize sequence numbers
2^2 (=0x04)	RST	Reset connection
2^3 (=0x08)	PSH	Push data
2^4 (=0x10)	ACK	Acknowledgement field value valid

tcpFlags	Flag	Description
2^5 (=0x20)	URG	Urgent pointer valid
2^6 (=0x40)	ECE	ECN-Echo
2^7 (=0x80)	CWR	Congestion Window Reduced flag is set

53.3.5 tcpAnomaly

The `tcpAnomaly` column is to be interpreted as follows:

tcpAnomaly	Description
`0x0001`	FIN-ACK flag
`0x0002`	SYN-ACK flag
`0x0004`	RST-ACK flag
`0x0008`	SYN-FIN flag, scan or malicious packet
`0x0010`	SYN-FIN-RST flag, potential malicious scan packet or channel
`0x0020`	FIN-RST flag, abnormal flow termination
`0x0040`	Null flag, potential NULL scan packet, or malicious channel
`0x0080`	XMas flag, potential Xmas scan packet, or malicious channel
`0x0100`	L4 option field corrupt or not acquired
`0x0200`	SYN retransmission
`0x0400`	Sequence Number retry
`0x0800`	Sequence Number out of order
`0x1000`	Sequence mess in flow order due to pcap packet loss
`0x2000`	Sequence number jump forward
`0x4000`	ACK number out of order
`0x8000`	Duplicate ACK

53.3.6 tcpOptions

The `tcpOptions` column is to be interpreted as follows:

tcpOptions	Description
2^0 (=`0x00000001`)	End of Option List
2^1 (=`0x00000002`)	No-Operation
2^2 (=`0x00000004`)	Maximum Segment Size
2^3 (=`0x00000008`)	Window Scale
2^4 (=`0x00000010`)	SACK Permitted
2^5 (=`0x00000020`)	SACK
2^6 (=`0x00000040`)	Echo (obsoleted by option 8)
2^7 (=`0x00000080`)	Echo Reply (obsoleted by option 8)
2^8 (=`0x00000100`)	Timestamps
2^9 (=`0x00000200`)	Partial Order Connection Permitted (obsolete)
2^{10} (=`0x00000400`)	Partial Order Service Profile (obsolete)
2^{11} (=`0x00000800`)	CC (obsolete)
2^{12} (=`0x00001000`)	CC.NEW (obsolete)
2^{13} (=`0x00002000`)	CC.ECHO (obsolete)
2^{14} (=`0x00004000`)	TCP Alternate Checksum Request (obsolete)
2^{15} (=`0x00008000`)	TCP Alternate Checksum Data (obsolete)

tcpOptions	Description
2^{16} (=0x00010000)	Skeeter
2^{17} (=0x00020000)	Bubba
2^{18} (=0x00040000)	Trailer Checksum Option
2^{19} (=0x00080000)	MD5 Signature Option (obsoleted by option 29)
2^{20} (=0x00100000)	SCPS Capabilities
2^{21} (=0x00200000)	Selective Negative Acknowledgements
2^{22} (=0x00400000)	Record Boundaries
2^{23} (=0x00800000)	Corruption experienced
2^{24} (=0x01000000)	SNAP
2^{25} (=0x02000000)	Unassigned (released 2000-12-18)
2^{26} (=0x04000000)	TCP Compression Filter
2^{27} (=0x08000000)	Quick-Start Response
2^{28} (=0x10000000)	User Timeout Option (also, other known unauthorized use)
2^{29} (=0x20000000)	TCP Authentication Option (TCP-AO)
2^{30} (=0x40000000)	Multipath TCP (MPTCP)
2^{31} (=0x80000000)	all options > 31

53.4 Packet File Output

In packet mode (`-s` option), the tcpFlags plugin outputs the following columns:

Column	Description	Flags
ipTOS	IP Type of Service	
ipID	IP ID	
ipIDDiff	IP ID diff	
ipFrag	IP fragment	
ipTTL	IP TTL	
ipHdrChkSum	IP header checksum	
ipCalChkSum	IP header computed checksum	
l4HdrChkSum	Layer 4 header checksum	
l4CalChkSum	Layer 4 header computed checksum	
ipFlags	IP flags	
ipOptLen	IP options length	
ipOpts	IP options	
seq	Sequence number	
ack	Acknowledgement number	
seqDiff	Sequence number diff	SEQ_ACK_NUM=1
ackDiff	Acknowledgement number diff	SEQ_ACK_NUM=1
seqPktLen	Sequence packet length	SEQ_ACK_NUM=1
ackPktLen	Acknowledgement packet length	SEQ_ACK_NUM=1
tcpFStat	TCP aggregated protocol flags (CWR, ACK, PSH, RST, SYN, FIN)	

Column	Description	Flags
tcpFlags	Flags	
tcpAnomaly	TCP aggregated header anomaly flags	
tcpWin	TCP window size	
tcpOptLen	TCP options length	
tcpOpts	TCP options	

53.5 Plugin Report Output

The aggregated ipFlags, tcpAnomaly and tcpWinSzThRt are reported.

53.6 References

- http://www.iana.org/assignments/ip-parameters
- http://www.iana.org/assignments/tcp-parameters/tcp-parameters.xml

54 tcpStates

54.1 Description

The tcpStates plugin tracks the actual state of a TCP connection, by analyzing the flags set in the packet header. The plugin recognizes and reports non-compliant behavior.

54.2 Configuration Flags

None.

54.3 Flow File Output

The tcpStates plugin outputs the following columns:

Column	Type	Description
tcpStates	H8	TCP state machine anomalies

54.3.1 tcpStates

The tcpStates column is to be interpreted as follows:

tcpStates	Description
0x01	Malformed connection establishment
0x02	Malformed teardown
0x04	Malformed flags during established connection
0x08	Packets detected after teardown
0x10	Packets detected after reset
0x40	Reset from sender
0x80	Potential evil behavior (scan)

54.3.2 Flow Timeouts

The tcpStates plugin also changes the timeout values of a flow according to its recognized state:

State	Description	Timeout (seconds)
New	Three way handshake is encountered	120
Established	Connection established	610
Closing	Hosts are about to close the connection	120
Closed	Connection closed	10
Reset	Connection reset encountered by one of hosts	0.1

54.3.3 Differences to the Host TCP State Machines

The plugin state machine (Figure 4) and the state machines usually implemented in hosts differ in some cases. Major differences are caused by the benevolence of the plugin. For example, if a connection has not been established in a correct

way, the plugin treats the connection as established, but sets the *malformed connection establishment* flag. The reasons for this benevolence are the following:

- A flow might have been started before invocation of Tranalyzer2.
- A flow did not finish before Tranalyzer2 terminated.
- Tranalyzer2 did not detect every packet of a connection, for example due to a router misconfiguration.
- Flows from malicious programs may show suspicious behavior.
- Packets may be lost **after** being captured by Tranalyzer2 but **before** they reached the opposite host.

54.4 Plugin Report Output

The aggregated tcpStates anomalies is reported.

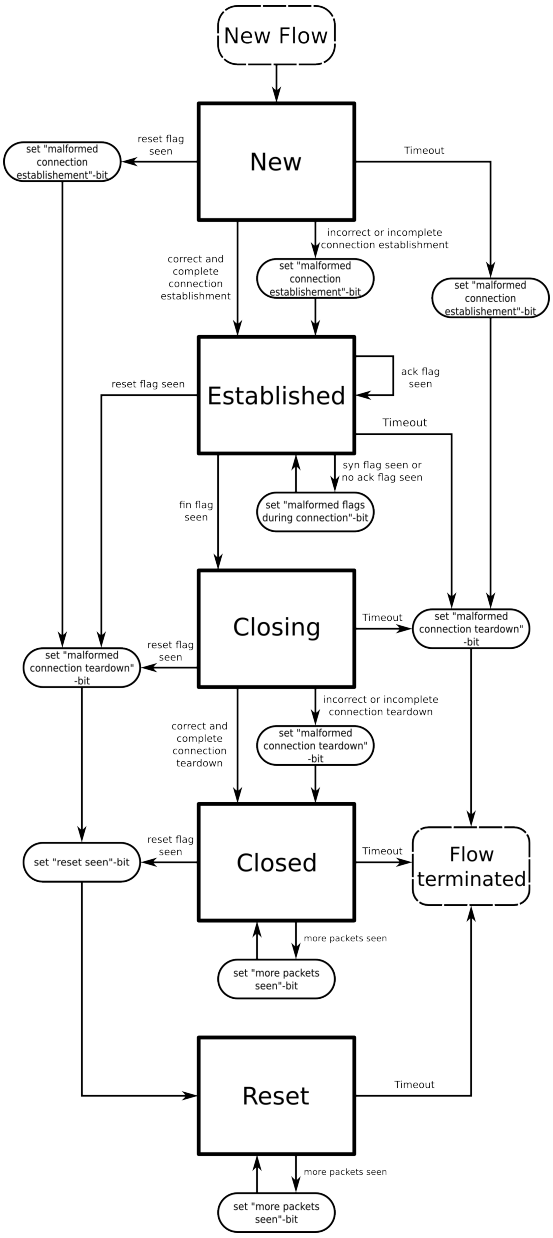


Figure 4: *State machine of the tcpState plugin*

55 telnetDecode

55.1 Description

The telnetDecode plugin analyses TELNET traffic and is capable to extract L7 content.

55.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
TEL_SAVE	0	Save content to TEL_F_PATH/TELFNAME
TEL_CMDC	0	output command codes
TEL_CMDS	1	output command human readable
TEL_OPTS	1	output options human readable
TEL_CMD_AGGR	1	Aggregate commands
TEL_OPT_AGGR	1	Aggregate options
TELCMDN	25	maximal command / flow
TELOPTN	25	maximal options / flow
TEL_F_PATH	"/tmp/TELFILES/"	Path for extracted content
TELFNAME	"telwurst"	file name

55.3 Flow File Output

The telnetDecode plugin outputs the following columns:

Column	Type	Description	Flags
telStat	H8	Status	
telCmdBF	H16	Commands	
telOptBF	H32	Options	
telTCCnt	U16	Total command count	
telTOCnt	U16	Total option count	
telCCnt	U16	Stored command count	TEL_CMDS=1||TEL_CMDC=1
telCmdC	RU8	Command codes	TEL_CMDC=1
telCmdS	RS	Command strings	TEL_CMDS=1
telOCnt	U16	Stored options count	TEL_OPTS=1
telOptS	RS	Option strings	TEL_OPTS=1

55.3.1 telStat

The telStat column is to be interpreted as follows:

telStat	Description
2^0 (=0x01)	TELNET port found
2^1 (=0x02)	—
2^2 (=0x04)	—

telStat	Description
2^3 (=0x08)	—
2^4 (=0x10)	—
2^5 (=0x20)	File open error: TEL_SAVE=1
2^6 (=0x40)	—
2^7 (=0x80)	—

55.3.2 telCmdBF

The telCmdBF column is to be interpreted as follows:

telCmdBF	Description
2^0 (=0x0001)	SE - End subNeg
2^1 (=0x0002)	NOP - No Op
2^2 (=0x0004)	Data Mark
2^3 (=0x0008)	Break
2^4 (=0x0010)	Int process
2^5 (=0x0020)	Abort output
2^6 (=0x0040)	Are You there?
2^7 (=0x0080)	Erase char

telCmdBF	Description
2^8 (=0x0100)	Erase line
2^9 (=0x0200)	Go ahead!
2^{10} (=0x0400)	SB - SubNeg
2^{11} (=0x0800)	WILL use
2^{12} (=0x1000)	WON'T use
2^{13} (=0x2000)	DO use
2^{14} (=0x4000)	DON'T use
2^{15} (=0x8000)	IAC

55.3.3 telOptBF

The telOptBF column is to be interpreted as follows:

telOptBF	Description
2^0 (=0x00000001)	Bin Xmit
2^1 (=0x00000002)	Echo Data
2^2 (=0x00000004)	Reconn
2^3 (=0x00000008)	Suppr GA
2^4 (=0x00000010)	Msg Sz
2^5 (=0x00000020)	Opt Stat
2^6 (=0x00000040)	Timing Mark
2^7 (=0x00000080)	R/C XmtEcho
2^8 (=0x00000100)	Line Width
2^9 (=0x00000200)	Page Length
2^{10} (=0x00000400)	CR Use
2^{11} (=0x00000800)	Horiz Tabs
2^{12} (=0x00001000)	Hor Tab Use
2^{13} (=0x00002000)	FF Use
2^{14} (=0x00004000)	Vert Tabs
2^{15} (=0x00008000)	Ver Tab Use

telOptBF	Description
2^{16} (=0x00010000)	Lf Use
2^{17} (=0x00020000)	Ext ASCII
2^{18} (=0x00040000)	Logout
2^{19} (=0x00080000)	Byte Macro
2^{20} (=0x00100000)	Data Term
2^{21} (=0x00200000)	SUPDUP
2^{22} (=0x00400000)	SUPDUP Outp
2^{23} (=0x00800000)	Send Locate
2^{24} (=0x01000000)	Term Type
2^{25} (=0x02000000)	End Record
2^{26} (=0x04000000)	TACACS ID
2^{27} (=0x08000000)	Output Mark
2^{28} (=0x10000000)	Term Loc
2^{29} (=0x20000000)	3270 Regime
2^{30} (=0x40000000)	X.3 PAD
2^{31} (=0x80000000)	Window Size

55.3.4 telCmdC and telCmdS

The `telCmdC` and `telCmdS` columns are to be interpreted as follows:

telCmdC	telCmdS	Description
0xf0	SE	Subnegotiation End
0xf1	NOP	No Operation
0xf2	DM	Data Mark
0xf3	BRK	Break
0xf4	IP	Interrupt Process
0xf5	AO	Abort Output
0xf6	AYT	Are You There
0xf7	EC	Erase Character
0xf8	EL	Erase Line
0xf9	GA	Go Ahead
0xfa	SB	Subnegotiation
0xfb	WILL	Will Perform
0xfc	WONT	Won't Perform
0xfd	DO	Do Perform
0xfe	DONT	Don't Perform
0xff	IAC	Interpret As Command

55.4 TODO

- fragmentation

56 tftpDecode

56.1 Description

The `tftpDecode` plugin analyses TFTP traffic. User defined compiler switches are in *tftpDecode.h*.

56.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
`TFTP_SAVE`	0	save content to FTP_F_PATH
`TFTP_MXNMLN`	15	maximal name length
`MAXCNM`	2	maximal length of command field
`FTP_F_PATH`	`"/tmp/TFTPFILES/"`	path for `TFTP_SAVE`

56.3 Flow File Output

The tftpDecode plugin outputs the following columns:

Column	Type	Description
`tftpStat`	H16	TFTP status bitfield
`tftPFlw`	U64	TFTP Parent Flow
`tftpOpCBF`	H8	TFTP OP Code Bit Field
`tftpErrCBF`	H8	TFTP Error Code Bit Field
`tftOpCNum`	U8	TFTP Number of OP Code
`tftpPNum`	U8	TFTP Number of parameters
`tftpOpC`	RSC	TFTP OP Codes
`tftpC`	RS	TFTP Parameters

56.3.1 tftpStat

The `tftpStat` column describes the errors encountered during the flow lifetime:

tftpStat	Name	Description
2^0 (=`0x0001`)	TFTPS_INIT	TFTP flow found
2^1 (=`0x0002`)	TFTPS_DRD	TFTP data read
2^2 (=`0x0004`)	TFTPS_DWD	TFTP data write
2^3 (=`0x0008`)	TFTP_FERR	file open error for TFTP_SAVE
2^4 (=`0x0010`)	TFTPS_BSERR	Error in block send sequence
2^5 (=`0x0020`)	TFTPS_BSAERR	Error in block ack sequence
2^6 (=`0x0040`)	TFTPS_PERR	Error or TFTP protocol error or not TFTP
2^7 (=`0x0080`)	TFTPS_OVFL	array overflow
2^8 (=`0x0100`)	—	—
2^9 (=`0x0200`)	—	—
2^{10} (=`0x0400`)	—	—

tftpStat	Name	Description
2^{11} (=0x0800)	TFTP_RW_PLNERR	Crafted packet or TFTP read/write parameter length error
2^{12} (=0x1000)	TFTPS_ACT	TFTP active
2^{13} (=0x2000)	TFTPS_PSV	TFTP passive
2^{14} (=0x4000)	—	—
2^{15} (=0x8000)	—	—

56.3.2 tftpOpCBF

The `tftpOpCBF` column describes the op code encountered during the flow lifetime:

tftpOpCBF	Name	Description
2^0 (=0x01)	TFTP_RRQ	1: Read request
2^1 (=0x02)	TFTP_WRQ	2: Write request
2^2 (=0x04)	TFTP_DATA	3: Read or write the next block of data
2^3 (=0x08)	TFTP_ACK	4: Acknowledgment
2^4 (=0x10)	TFTP_ERR	5: Error message
2^5 (=0x20)	TFTP_OACK	6: Option acknowledgment
2^6 (=0x40)	—	—
2^7 (=0x80)	—	—

56.3.3 tftpErrCBF

The `tftpErrCBF` column describes the error code (if op code TFTP_ERR encountered during the flow lifetime):

tftpErrCBF	Name	Description
(=0x00)	TFTP_NOERR	0: No Error
2^0 (=0x01)	TFTP_FLNFND	1: File not found
2^1 (=0x02)	TFTP_ACCVLT	2: Access violation
2^2 (=0x04)	TFTP_DSKFLL	3: Disk full or allocation exceeded
2^3 (=0x08)	TFTP_ILGLOP	4: Illegal TFTP operation
2^4 (=0x10)	TFTP_UKWNID	5: Unknown transfer ID
2^5 (=0x20)	TFTP_FLEXST	6: File already exists
2^6 (=0x40)	TFTP_NOSUSR	7: No such user
2^7 (=0x80)	TFTP_TRMOPN	8: Terminate transfer due to option negotiation

56.4 TODO

- fragmentation
- reply address extraction

57 tp0f

57.1 Description

The tp0f plugin classifies IP addresses according to OS type and version. It uses initial TTL and window size and can also use the rules from p0f. In order to label non-TCP flows, the plugin can store a hash of already classified IP addresses.

57.1.1 Required Files

If `TP0FRULES=1`, then the file `tp0fL34.txt` is required.

57.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description
`TP0FRULES`	1	0: standard OS guessing; 1: OS guessing and p0f L3/4 rules
`TP0FHSH`	1	0: no IP hash; 1: IP hash to recognize IP already classified
`TP0FRC`	0	0: only human readable; 1: tp0f rule and classifier numbers
`TP0FL34FILE`	`"tp0fL34.txt"`	file containing converted L3/4 rules

In *tp0flist.h*:

Name	Default	Description
`MAXLINELN`	4096	maximal line input buffer size for *tp0fL34.txt*
`TCPOPTMAX`	40	maximal TCP option byted codes being stored and processed

57.3 Flow File Output

The p0f plugin outputs the following columns:

Column	Type	Description
`tp0fStat`	H8	status
`tp0fDis`	U8	initial ttl distance
`tp0fRN`	U16	rule number that triggered
`tp0fClass`	U8	OS class of rule file
`tp0fProg`	U8	Program category of rule file
`tp0fVer`	U8	version category of rule file
`tp0fClName`	SC	OS class name
`tp0fPrName`	SC	OS/Program name
`tp0fVerName`	SC	OS/Program version name

57.3.1 tp0fStat

The `tp0fStat` column is to be interpreted as follows:

tp0fStat	Description
0x01	SYN tp0f rule fired
0x02	SYN-ACK tp0f rule fired
0x04	—
0x08	—
0x10	—
0x20	—
0x40	IP already seen by tp0f
0x80	TCP option length or content corrupt

57.4 Plugin Report Output

The number of packets which fired a tp0f rule is reported.

57.5 TODO

- Integrate TLS rules
- Integrate HTTP rules

57.6 References

- http://www.netresec.com/?page=Blog&month=2011-11&post=Passive-OS-Fingerprinting
- http://lcamtuf.coredump.cx/p0f3/

58 txtSink

58.1 Description

The txtSink plugin provides human readable text output which can be saved in a file `PREFIX_flows.txt`, where `PREFIX` is provided via the `-w` option. The generated output contains a textual representation of all plugins results. Each line in the file represents one flow. The different output statistics of the plugins are separated by a tab character to provide better post-processing with command line scripts or statistical toolsets.

58.2 Dependencies

58.2.1 External Libraries

If gzip compression is activated (`GZ_COMPRESS=1`), then **zlib** must be installed.

Kali/Ubuntu: `sudo apt-get install zlib1g-dev`

Arch: `sudo pacman -S zlib`

Fedora/Red Hat: `sudo yum install zlib-devel`

Gentoo: `sudo emerge zlib`

OpenSUSE: `sudo zypper install zlib-devel`

Mac OS X: `brew install zlib`[11]

58.3 Configuration Flags

The configuration flags for the txtSink plugins are separated in two files.

58.3.1 txtSink.h

Name	Default	Description
`TFS_SPLIT`	1	Split the output file (Tranalyzer `-W` option)
`TFS_PRI_HDR`	1	Print a row with column names at the start of the flow file
`TFS_HDR_FILE`	1	Generate a separate header file (Section 58.4.1)
`TFS_PRI_HDR_FW`	0	Print header in every output fragment (Tranalyzer `-W` option)
`GZ_COMPRESS`	0	Compress the output (gzip)

The default suffix used for the flow file is `_flows.txt` and `_headers.txt` for the header file. Both suffix can be configured using `FLOWS_TXT_SUFFIX` and `HEADER_SUFFIX` respectively.

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

58.3.2 bin2txt.h

bin2txt.h controls the conversion from internal binary format to standard text output.

Variable	Default	Description
HEX_CAPITAL	0	Hex number representation: 0: lower case, 1: upper case
IP4_NORMALIZE	0	IPv4 addresses representation: 0: normal, 1: normalized (padded with 0)
IP6_COMPRESS	1	IPv6 addresses representation: 1: compressed, 0: full 128 bit length
TFS_EXTENDED_HEADER	0	Whether or not to print an extended header in the flow file (number of rows, columns, columns type)
B2T_LOCALTIME	0	Time representation: 0: UTC, 1: localtime
B2T_TIME_IN_MICRO_SECS	1	Time precision: 0: nanosecs, 1: microsecs
HDR_CHR	"%"	start character of comments in flow file
SEP_CHR	"\t"	character to use to separate the columns in the flow file

58.4 Additional Output

58.4.1 Header File

The header file PREFIX_headers.txt describes the columns of the flow file and provides some additional information, such as plugins loaded and PCAP file or interface used, as depicted below. The default suffix used for the header file is _headers.txt. This suffix can be configured using HEADER_SUFFIX.

```
# Header file for flow file: PREFIX_flows.txt
# Generated from: /home/test/file.pcap
#
# 666;03.03.2016_19:04:55;hostname;Linux;4.2.0-30-generic;#36-Ubuntu SMP Fri Feb 26 00:58:07
    UTC 2016;x86_64
#
# Plugins loaded:
# 00: protoStats, version 0.6.0
# 01: basicFlow, version 0.6.0
# 02: macRecorder, version 0.6.0
# 03: portClassifier, version 0.5.8
# 04: basicStats, version 0.6.1
# 05: tcpFlags, version 0.6.0
# 06: tcpStates, version 0.5.8
# 07: icmpDecode, version 0.6.0
# 08: connectionCounter, version 0.6.0
# 09: txtSink, version 0.5.8
#
# Col No.   Type        Name
1           24:N        Flow direction
2           10:N        Flow Index
3           15:N        Flow Status
4           25:N        System time of first packet
5           25:N        System time of last packet
6           25:N        Flow duration
7           8:R         Ether VlanID
8           28:N        Source IPv4 address
9           15:N        Subnet number of source IPv4
10          8:N         Source port
11          28:N        Destination IP4 address
12          15:N        Subnet number of destination IP
```

```
13          8:N         Destination port
14          7:N         Layer 4 protocol
15          9:N         Number of distinct Source/Destination MAC addresses pairs
16          27_27_10:R  Source MAC address, destination MAC address, number of packets of MAC
    address combination
17          30_30:R Source MAC manufacturer, destination MAC manufacturer
...
```

The column number can be used, e.g., with `awk` to query a given column. For example, to extract all ICMP flows (layer 4 protocol equals 1) from a flow file:

```
awk -F'\t' '$14 == 1' PREFIX_flows.txt
```

The second column indicates the type of the column (see table below). If the value is repetitive, the type is postfixed with `:R`. Repetitive values can occur any number of times (from 0 to *N*). Each repetition is separated by a semicolon. The `'_'` indicates a compound, i.e., a value containing 2 or more subvalues.

#	Name	Description
1	I8	int8
2	I16	int16
3	I32	int32
4	I64	int64
5	I128	int128
6	I256	int256
7	U8	uint8
8	U16	uint16
9	U32	uint32
10	U64	uint64

#	Name	Description
11	U128	uint128
12	U256	uint256
13	H8	hex8
14	H16	hex16
15	H32	hex32
16	H64	hex64
17	H128	hex128
18	H256	hex256
19	F	float
20	D	double

#	Name	Description
21	LD	long double
22	C	char
23	S	string
24	C	flow direction[12]
25	TS	timestamp[13]
26	U64.U32	duration
27	MAC	mac address
29	IP4	IPv4 address
29	IP6	IPv6 address
30	IPX	IPv4 or 6 address
31	SC	string class[14]

[12] `A`: client→server, `B`: server→client
[13] U64.U32/S (See `B2T_TIMESTR` in bin2txt.h)
[14] string without quotes

59 voipDetector

59.1 Description

The idea of this plugin is to identify SIP, RTP and RTCP flows independently of each other, so that also non standard traffic can be detected. Moreover certain QoS values are extracted.

59.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Variable	Default	Description	Flags
VOIP_ANALEN	1	1: additional check report len against payload length 0: only ssrc check	
VOIP_V_SAVE	0	save rtp content to VOIP_RM_DIR	
VOIP_RM_DIR	0	rm RTP content directory	VOIP_V_SAVE=1
VOIP_PLDOFF	0	offset to payload pointer to save content	VOIP_V_SAVE=1
SIPNMMAX	40	maximal sip caller name length in flow file	
VOIP_PATH	"/tmp/"	default path of content directory	
VOIP_FNAME	"eier"	default content file name prefix	

59.3 Flow File Output

The voipDetector plugin outputs the following columns:

Column	Type	Description
voipStat	H16	Status
voipID	H32	RTP/RTCP ID
voipSRCnt	U8	RTP SID/RTCP record count
voipTyp	U8	RTP/RTCP type
voipPMCnt	U32	RTP packet miss count
voipPMr	F	RTP packet miss ratio
voipSIPStatCnt	U8	SIP stat count
voipSIPReqCnt	U8	SIP request count
voipSIPCID	S	SIP Call ID
voipSIPStat	R(U16)	SIP stat
voipSIPReq	R(S)	SIP request
voipTPCnt	U32	RTCP cumulated transmitter packet count
voipTBCnt	U32	RTCP cumulated transmitter byte count
voipCPMCnt	U32	RTCP cumulated packet miss count
voipMaxIAT	U32	RTCP maximal Inter Arrival Time

59.3.1 voipStat

The `voipStat` column is to be interpreted as follows:

voipStat	Name	Description
2^0 (=`0x0001`)	RTP	RTP detected
2^1 (=`0x0002`)	RTCP	RTCP detected
2^2 (=`0x0004`)	SIP	SIP detected
2^3 (=`0x0008`)	STUN	STUN present
2^4 (=`0x0010`)	X	RTP: extension header
2^5 (=`0x0020`)	P	RTP: padding present
2^6 (=`0x0040`)	-	-
2^7 (=`0x0080`)	M	RTP: data marker set
2^8 (=`0x0100`)	WROP	RTP: content write operation
2^9 (=`0x0200`)	-	-
2^{10} (=`0x0400`)	-	-
2^{11} (=`0x0800`)	-	-
2^{12} (=`0x1000`)	PKTLSS	RTP: packet loss detected
2^{13} (=`0x2000`)	RTPNFRM	RTP: new frame header flag
2^{14} (=`0x4000`)	-	-
2^{15} (=`0x8000`)	-	-

59.4 TODO

- Skype
- Google Talk

60 vrrpDecode

60.1 Description

The vrrpDecode plugin analyzes Virtual Router Redundancy Protocol (VRRP) traffic.

60.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Name	Default	Description	Flags
VRRP_NUM_VRID	5	number of unique virtual router ID to store	
VRRP_NUM_IP	25	number of unique IPs to store	
VRRP_RT	0	Whether (1) or not (0) to output routing tables	
VRRP_SUFFIX	"_vrrp.txt"	Suffix for routing tables file	VRRP_RT=1

60.3 Flow File Output

The vrrpDecode plugin outputs the following columns:

Column	Type	Description
vrrpStat	H16	Status
vrrpVer	H8	Version
vrrpType	H8	Type
vrrpVRIDCnt	U32	Virtual router ID count
vrrpVRID	RU8	Virtual router ID
vrrpMinPri	U8	Minimum priority
vrrpMaxPri	U8	Maximum priority
vrrpMinAdvInt	U8	Minimum advertisement interval [s]
vrrpMaxAdvInt	U8	Maximum advertisement interval [s]
vrrpAuthType	H8	Autentication type
vrrpAuth	SC	Authentication string
vrrpIPCnt	U32	IP address count
vrrpIP	R(IP)	IP addresses

60.3.1 vrrpStat

The vrrpStat column is to be interpreted as follows:

vrrpStat	Description
0x0001	flow is VRRP
0x0002	invalid version
0x0004	invalid type
0x0008	invalid checksum
0x0010	invalid TTL (should be 255)
0x0020	invalid destination IP (should be 224.0.0.18)

vrrpStat	Description
0x0040	invalid destination MAC (should be 00:00:5e:00:01:routerID)
0x0100	Virtual Router ID list truncated... increase VRRP_NUM_VRID
0x0200	IP list truncated... increase VRRP_NUM_IP
0x4000	Packet snapped
0x8000	Malformed packet... covert channel?

60.3.2 vrrpVer

The vrrpVer column is to be interpreted as follows:

vrrpVer	Description
0x04	VRRP v2
0x08	VRRP v3

60.3.3 vrrpType

The vrrpType column is to be interpreted as follows:

vrrpType	Description
0x01	Advertisement

60.3.4 vrrpAuthType

The vrrpAuthType column is to be interpreted as follows:

vrrpAuthType	Description
0x01	No authentication
0x02	Simple text password
0x04	IP Authentication Header

60.4 Additional Output

Non-standard output:

- PREFIX_vrrp.txt: VRRP routing tables

The routing tables contain the following columns:

Name	Description
VirtualRtrID	Virtual router ID
Priority	Priority
SkewTime[s]	Skew time (seconds)
MasterDownInterval[s]	Master down interval (seconds)
AddrCount	Number of addresses

Name	Description
Addresses	List of addresses
Version	VRRP version
Type	Message type
AdverInt[s]	Advertisement interval
AuthType	Authentication type
AuthString	Authentication string
Checksum	Stored checksum
CalcChecksum	Calculated checksum
flowIndex	Flow index

60.5 Plugin Report Output

The number of VRRP v2 and v3 packets is reported.

60.6 Post-Processing

The routing tables can be pruned by using the following command:

```
sort -u PREFIX_vrrp.txt > PREFIX_vrrp_pruned.txt
```

61 wavelet

61.1 Description

This plugin calculates the Daubechies wavelet transformation of the IP packet length variable in the packet structure, or the inter arrival distance of packets (IAT). The wavelet plugin requires no dependencies and produces only output to the flow file. User defined compiler switches in *define_global.h* produce optimized code for the specific application.

61.2 Configuration Flags

The following flags can be used to control the output of the plugin:

Variable	Default	Description
WAVELET_IAT	0	analysis of 0: pktlen, 1:IAT
WAVELET_SIG	0	print signal
WAVELET_PREC	0	precision 0: float, 1: double
WAVELET_THRES	8	Min number of packets for analysis
WAVELET_LEVEL	3	Wavelet decomposition level
WAVELET_EXTMODE	ZPD	Extension Mode: NON, SYM, ZPD
WAVELET_TYPE	DB3	Mother Wavelet: Daubechies DB1 - DB4
WAVELET_MAX_PKT	40	Maximal # of selected pkts

61.3 Flow File Output

The wavelet plugin outputs the following columns:

Name	Type	Description
waveNumPnts	U16	Number of points
waveSig	RF/D	Packet length / IAT signal
waveNumLvl	U32	Number of wavelet levels
waveCoefDetail	RF/D	Wavelet detail coefficients
waveCoefApprox	RF/D	Wavelet approximation coefficients

62 scripts

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

62.1 b64ex

Extracts all HTTP, EMAIL, FTP, TFTP etc base64 encoded content extracted from T2 und /tmp. 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`

62.2 flowstat

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

62.3 statGplt

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

62.4 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>

Optional arguments:

    -f findex        Flow index to extract, default: all flows
    -d 0|1           Flow Direction: 0, 1; default both
    -t               No Time: 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]

    -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.

62.5 fpsEst

This script takes the output file of `fpsGplt` and calculates the jumps in IAT to allow the user to choose an appropriate `MINIAT(S/U)` in nFrstPkts plugin.

```
fpsEst flowfile_nps.txt
```

62.6 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`

62.7 new_plugin

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

62.8 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 directoy, 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`

62.9 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). It can output the top or bottom *N* protocols or only those with at least a given percentage:

- list all the options: `protStat --help`
- 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`

62.10 rrdmonitor

Stores Tranalyzer monitoring output into a RRD database.

62.11 rrdplot

Uses the RRD database generated by `rrdmonitor` to monitor and plot various values, e.g., number of flows.

62.12 segvtrack

If the processing of a pcap file causes a segmentation fault, this script can be used to locate the packets which caused the error. It works by repetitively splitting the file in half until neither half causes a segmentation fault. Its usage is as follows:

```
segvtrack file.pcap
```

Note that you might need to change the path to the Tranalyzer binary by editing the `T2` variable at line 5 of the script.

62.13 t2_aliases

Set of aliases for Tranalyzer.

62.13.1 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, e.g., `tcpFlags` 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 access tawk , 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 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`

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

gt2
Shortcut to run Tranalyzer in gdb 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`

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

rrdmonitor
Shortcut to run rrdmonitor from anywhere, e.g., `t2 -i eth0 | rrdmonitor`

rrdplot
Shortcut to run rrdplot from anywhere, e.g., `rrdplot V4Pkts V6Pkts`

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 -t2`

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

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

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

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

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

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

62.13.2 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/tranalyzer2-0.8.3`):

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

62.13.3 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`

62.14 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:

T2 on the other hand has also to be configured. To preserve simplicity the unused SIGSYS 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:

`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

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 300: *t2alive script configuration*

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 301: *T2 configuration for t2alive mode*

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.

62.15 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`

62.16 t2conf

Use `t2conf` to build, configure, activate and deactivate Tranalyzer plugins or use the `t2plconf` script provided with all the plugins to configure individual plugins as follows:

- `cd $T2HOME/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.

62.16.1 Dependencies

The t2conf 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 62.16.3). 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 7 of the t2conf script and at line 66 of the t2plconf script.

62.16.2 t2confrc

Set of predefined settings for `t2conf`.

62.16.3 Installation

The easiest way to install `t2conf` and its dependencies is to use the provided `install.sh` script: `./install.sh --help`
1Y

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. To use the predefined settings, copy *t2confrc* to `~/.tranalyzer/plugins/`.

62.16.4 Usage

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

62.16.5 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 similar to *t2confrc*:

- 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[15]

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

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`

62.17 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.

62.17.1 Dependencies

This script requires **inotify-tools**:

Arch: `sudo pacman -S inotify-tools`

Fedora: `sudo yum install inotify-tools`

Gentoo: `sudo emerge inotify-tools`

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

62.17.2 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.

62.18 t2doc

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

62.19 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`

62.19.1 Required Plugins

- basicFlow
- basicStats
- txtSink

62.19.2 Optional Plugins

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

62.20 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 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`

Mac OSX: `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`
- `t2plot file_with_two_or_three_columns.txt`
- `t2plot -o "26:28" file_with_many_columns.txt`
- `t2plot -o "numBytesSnt:numBytesRcvd" file_with_many_columns.txt`

62.21 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 comfirmation 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.

62.22 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_ethVlanID`.
- 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 5.

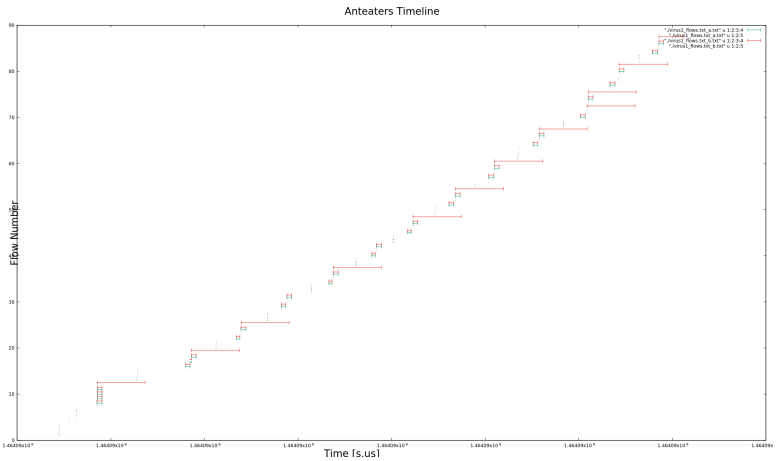


Figure 5: *T2 timeline flow plot*

62.23 t2utils.sh

Collection of bash functions and variables.

62.23.1 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
```

62.23.2 Colors

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

Variable	Description	Example
`BLUE`	Set the color to blue	`printf "${BLUE}message${NOCOLOR}\n"`
`GREEN`	Set the color to green	`echo -e "${GREEN}message${NOCOLOR}"`
`ORANGE`	Set the color to orange	`printf "${ORANGE}${1}${NOCOLOR}\n" "message"`
`RED`	Set the color to red	`echo -e "${RED}$1${NOCOLOR}" "message"`
`BLUE_BOLD`	Set the color to blue bold	`printf "${BLUE_BOLD}message${NOCOLOR}\n"`
`GREEN_BOLD`	Set the color to green bold	`echo -e "${GREEN_BOLD}message${NOCOLOR}"`
`ORANGE_BOLD`	Set the color to orange bold	`printf "${ORANGE_BOLD}${1}${NOCOLOR}\n" "message"`
`RED_BOLD`	Set the color to red bold	`echo -e "${RED_BOLD}$1${NOCOLOR}" "message"`
`BOLD`	Set the font to bold	`echo -e "${BOLD}$1${NOCOLOR}" "message"`
`NOCOLOR`	Reset the color	`printf "${RED}message$NOCOLOR\n"`

62.23.3 Folders

Variable	Description	Example
`SHOME`	Points to the folder where the script resides	For `basicFlow/utils/subconv`, `SHOME` is `$T2PLHOME/basicFlow/utils`
`T2HOME`	Points to the root folder of Tranalyzer	`$T2HOME/scripts/new_plugin`
`T2PLHOME`	Points to the root folder of Tranalyzer plugins	`cd $T2PLHOME/t2PSkel`

62.23.4 Programs

Variable	Program	Example
`AWK`	`gawk`	`$AWK '{ print }' file`
`AWKF`	`gawk -F'\t' -v OFS='\t'`	`"${AWKF[@]}" '{ print } file'`
`OPEN`	`xdg-open` (Linux), `open` (MacOS)	`$OPEN file.pdf`
`READLINK`	`readlink` (Linux) / `greadlink` (MacOS)	`$READLINK file`
`SED`	`sed` (Linux) / `gsed` (MacOS)	`$SED 's/ /_/g' «< "$str"`
`T2`	`$T2HOME/tranalyzer2/src/tranalyzer`	`$T2 -r file.pcap`
`T2BUILD`	`$T2HOME/autogen.sh`	`$T2BUILD tranalyzer2`
`T2CONF`	`$T2HOME/scripts/t2conf/t2conf`	`$T2CONF -D SCTP_ACTIVATE=1 tranalyzer2`
`TAWK`	`$T2HOME/scripts/tawk/tawk`	`$TAWK '{ print tuple4() } file'`

62.23.5 Functions

Function	Description
`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

Function	Description
`printwrn "msg"`	print a warning message (orange) with a newline
`check_dependency "bin" "pkg"`	check whether a dependency exists (Linux/MacOS)
`check_dependency_linux "bin" "pkg"`	check whether a dependency exists (Linux)
`check_dependency_osx "bin" "pkg"`	check whether a dependency exists (MacOS)
`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`
`get_nproc`	return the number of processing units available
`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 the next argument exists and is not an option
`validate_next_arg_exists "curr" "next"`	check whether the next argument exists
`validate_next_dir "curr" "next"`	check whether the next argument exists and is a directory
`validate_next_file "curr" "next"`	check whether the next argument exists and is a regular file
`validate_next_pcap "curr" "next"`	check whether the next argument exists and is a PCAP file
`validate_next_num "curr" "next"`	check whether the next argument exists and is a positive integer
`validate_next_int "curr" "next"`	check whether the next argument exists and is an integer
`validate_next_float "curr" "next"`	check whether the next argument exists and is a float
`arg_is_option "arg"`	check whether `arg` exists and is an option (starts with -)
`abort_missing_arg "option"`	print a message about a missing argument and exit with status 1
`abort_option_unknown "option"`	print a message about an unknown option and exit with status 1
`abort_required_file`	print a message about a missing required file and exit with status 1
`abort_with_help`	print a message explaining how to get help and exit with status 1

62.24 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.

62.25 t2wizard

Launch several instances of Tranalyzer in the background, each with its own list of plugins (Tranalyzer must be configured to use a plugin loading list (*tranalyzer2/src/loadPlugins.h:24*: `USE_PLLIST > 0`). The script is interactive and will prompt for the required information. To see all the options available, run `t2wizard --help`. To use it, run `t2wizard -r file.pcap` or `t2wizard -R pcap_list.txt`.

62.26 topNStat

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

62.27 vc.c

Calculates entropy based features for a T2 column in a flow file or the packet file, selected by awk, tawk or cut, moreover it decodes the `'%'` HTTP notation for URLs.
Compile: `gcc vc.c -lm -o vc`
Example: extract url in position 26 and feed it into vc: `cut -f 26 file_flows.txt | ./vc`
Output on commandline:

```
...
5,45,17,4,0,9,0,0    1.000000    0.000000    0.549026    80    16.221350    0.342250
        "/hphotos-ak-snc4/hs693.snc4/63362_476428124179_624129179_6849488_4409532_n.jpg"
...
```

63 PDF Report Generation from PCAP using t2fm

63.1 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.

63.2 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[16] (make sure to replace `$T2HOME` with the actual path, e.g., `$HOME/tranalyzer2-0.7.0lm1/trunk`):

```
# $HOME/.bashrc

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

63.2.1 Required plugins

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

- basicFlow
- basicStats
- txtSink

63.2.2 Optional plugins

The following plugins are optional:

- arpDecode
- dnsDecode
- geoip
- pwX
- sshDecode
- sslDecode
- httpSniffer , configured as follows[17]:
  - `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.

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

63.2.3 Packages

The following packages are required to build the PDF:

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

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

For simplicity, this tutorial assumes the user wants a complete report, i.e., requires all of the optional plugins.

1. Make sure all the plugins are configured as described in Section 63.2
2. Build Tranalyzer and the plugins [18]:
   `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`:
   `evince file.pdf`

63.4 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 with the following steps:

1. Follow point 1 and 2 from Section 63.3
2. Run Tranalyzer on a pcap file as follows:
   `t2 -r file.pcap -w out`
3. The previous command should have created the following files:
   `out_headers.txt`
   `out_flows.txt`
4. Run the `t2fm` script on the flow file generated previously:
   `t2fm -F out_flows.txt`

63.5 Step-by-Step Instructions (MongoDB / PostgreSQL to PDF)

If the 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 63.3[19]
2. Build the mongoSink or psqlSink plugin:
   - **mongoDB:** `t2build mongoSink`

[18] Hint: use the tab completion to avoid typing the full name of all the plugins: `t2build tr<tab> ... ht<tab> ...`
[19] `HTTP_SAVE_*` do not need to be set as EXE downloads detection is currently not implemented in the DB backends

- **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:
   - **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., `2018-10-01 12:34:56.912345+0100`. Note that only the required fields must be specified, e.g., `2018-09-01` is equivalent to `2018-09-01 00:00:00.000000`. For example, to generate a report from the 1st of September to the 11. of October 2018 at 14:59 from a PostgreSQL database, run the following command: `t2fm -p tranalyzer -T "2018-09-01" "2018-10-11 14:59"`

63.6 Conclusion

This tutorial has presented how `t2fm` can be used to create a PDF report summarising the traffic contained in a PCAP file. 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`.

64 tawk

64.1 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.

64.2 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`[20]

64.3 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/tranalyzer2-0.8.3`):

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

64.3.1 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.

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

64.4 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. 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`.

64.5 `-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'`.

64.6 Related Utilities

64.6.1 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`

64.6.2 lsx

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

64.6.3 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`

64.6.4 tcol

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

64.7 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
ip()	Returns true if the flow contains IPv4 or IPv6 traffic
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

Function	Description
ip2num(ip)	Converts an IP address to a number
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 hexadecimal numbers val1 and val2 are equal
bitsallset(val,mask)	Returns true if all the bits set in mask are also set in val

Function	Description
bitsanyset(val,mask)	Returns true if one of the bits set in mask is also set in val
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
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(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 printed with print_max([c])
abs(v)	Returns the absolute value of v
min2(a,b)	Returns the minimum value between a and b
min3(a,b,c)	Returns the minimum value between a, b and c
max2(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.

Function	Description
	sep can be used to change the separator character (default: ";")
t2sort(col[,num[,type]])	Sorts the file according to col. 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

64.8 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,

Function	Description
	If num < 0 returns the bottom num results.
dnsZT()	Returns all flows where a DNS zone transfer was performed.
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, ...).

64.9 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

Function	Description
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 package
oline()	nfdump line output format (-o line)
olong()	nfdump long output format (-o long)
oextended()	nfdump extended output format (-o extended)

64.10 t2custom

Copy your own functions in this folder. Refer to Section 64.11 for more details on how to write a tawk function. To have your functions automatically loaded, include them in the file t2custom/t2custom.load.

64.11 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
    }
}
```

64.12 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`

64.13 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 64.13.1.

64.13.1 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.

64.13.2 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
```

64.14 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.

64.14.1 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`.

64.14.2 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
```

64.15 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
    ```

64.16 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`.

64.17 FAQ

64.17.1 Can I use tawk with non Tranalyzer files?

Yes, refer to Section 64.13.

64.17.2 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 64.13.1 for more details.

64.17.3 Can I use tawk with files without column names?

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

64.17.4 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'`

64.17.5 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`

64.17.6 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`

64.17.7 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`

64.17.8 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'`

64.17.9 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 64.5 for more details.

64.17.10 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.

64.17.11 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 64.5 for more details.

64.17.12 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.

64.17.13 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).

A Creating a Custom Plugin

A plugin is a shared library file comprising of special functionality. Tranalyzer2 dynamically loads these shared libraries at runtime from the *~/.tranalyzer/plugins* directory in the user's home folder. Therefore Tranalyzer2 is available for users if being installed in the */usr/local/bin* directory while the plugins are user dependent. To develop a plugin it is strongly recommended that the user utilizes our special "new_plugin" script. This script uses the plugin skeleton "t2PSkel" to create a new custom named plugin. It is available via SVN from the Tranalyzer repository under the *scripts/* folder. The script copies only the required files. Therefore it is recommended to upload the newly created folder to a SVN/GIT repository before running `./autogen.sh` (alternatively, `./autogen.sh -c` can be used to clean up automatically generated files that should not be committed). The skeleton contains a header and a source file comprising of all mandatory and optional functions as well as a small `HOWTO` file and a script to build and move a shared library to the plugins folder.

A.1 Plugin Name

Plugin names should be kept short, start with a lowercase letter and only contain characters in the following ranges: a–z, A–Z, 0–9. In addition, each "word" should start with an uppercase letter, e.g., `pluginName`.

A.2 Plugin Number

The plugin number (or order) influences when a plugin is to be loaded (useful if a plugin depends on another one). This number should consist of three digits and be unique. The plugin orders used in your Tranalyzer installation can be listed with `./scripts/pne`. As a rule of thumb, numbers greater than 900 should be kept for sink (output) plugins and numbers smaller than 10 for global plugins.

plugin range	description
000 - 099	global
100 - 199	basic L2/3/4 plugins
200 - 299	service and routing
300 - 699	L7 protocols
700 - 799	Math and statistics
800 - 899	classifier and AI
900 - 999	output

A.3 Plugin Creation

To create a new plugin named *pluginName* with plugin order 123, run the following command from Tranalyzer's root, i.e., `trunk` folder:

```
./scripts/new_plugin pluginName 123
```

If no plugin number is provided, then the script will choose a random one that is not used by any other plugin.

A.3.1 autogen.sh

The `autogen.sh` script provides the `EXTRAFILES` variable, which is used to list extra files, such as lists of subnets, protocols, services, databases or blacklists, that the plugin needs in order to run. The files listed in this variable are automatically copied into the Tranalyzer plugin folder.

```
EXTRAFILES=(dependency1 dependency2)
```

The `CFLAGS` variable in `autogen.sh` can be used if a plugin requires specific libraries, compilation or linking flags, e.g., `CFLAGS="-lzip"`. In such a case, the `DEPS` variable can be used to list the dependencies, e.g., `DEPS="libzip"`.

A.4 Compilation

The plugin can then be compiled by typing `./autogen.sh`. For a complete list of options, run `./autogen.sh -h`

A.5 Plugin Structure

All plugins have the same global structures, namely, a comment describing the license of the plugin, e.g., GPLv2+, some includes, followed by the declaration of variables and functions. This section discusses the Tranalyzer callbacks which follows the elements already mentionned. Note that all the callbacks are optional, but a plugin **MUST** call one of the initialization macros.

First, a plugin MUST have one the following declarations:

- `T2_PLUGIN_INIT(name, version, t2_v_major, t2_v_minor)`
- `T2_PLUGIN_INIT_WITH_DEPS(name, version, t2_v_major, t2_v_minor, deps)`

For example, to initialize `myPlugin`:

```
T2_PLUGIN_INIT_WITH_DEPS("myPlugin", "0.8.3", 0, 8, "tcpFlags,basicStats")
```

The available callbacks are:

- `void initialize()`
- `binary_value_t *printHeader()`
- `void onFlowGenerated(packet_t *packet, unsigned long flowIndex)`
- `void claimLayer2Information(packet_t *packet, unsigned long flowIndex)`
- `void claimLayer3Information(packet_t *packet)` [Deprecated]
- `void claimLayer4Information(packet_t *packet, unsigned long flowIndex)`
- `void onFlowTerminate(unsigned long flowIndex)`
- `void pluginReport(FILE *stream)`
- `void onApplicationTerminate()`
- `void bufferToSink(outputBuffer_t *buffer)` [Sink (output) plugins only]

The following callbacks offer more advanced capabilities:

- `void t2BusCallback(uint32_t status)` [Not implemented]
- `void monitoring(FILE *stream, uint8_t state)`
- `void saveState(FILE *stream)`
- `void restoreState(char *str)`

A.5.1 `void initialize()`

This function is called before processing any packet.

A.5.2 `binary_value_t *printHeader()`

This function is used to describe the columns output by the plugin Refer to Section A.7 and the `BV_APPEND` macros.

A.5.3 `void onFlowGenerated(packet_t *packet, unsigned long flowIndex)`

This function is called every time a new flow is created.

A.5.4 `void claimLayer2Information(packet_t *packet, unsigned long flowIndex)`

This function is called for every packet with a layer 2. If `flowIndex` is `HASHTABLE_ENTRY_NOT_FOUND`, this means the packet also has a layer 4 and thus a call to `claimLayer4Information()` will follow.

A.5.5 `void claimLayer3Information(packet_t *packet)`

This function is called for every packet with a layer 3.

A.5.6 `void claimLayer4Information(packet_t *packet, unsigned long flowIndex)`

This function is called for every packet with a layer 4.

A.5.7 `void onFlowTerminate(unsigned long flowIndex)`

This function is called once a flow is terminated. Output all the statistics for the flow here. Refer to Section A.7 and the `OUTBUF_APPEND` macros.

A.5.8 `void t2BusCallback(uint32_t status)`

Currently not implemented.

A.5.9 `void monitoring(FILE *stream, uint8_t state)`

This function is used to report information regarding the plugin at regular interval or when a `USR1` signal is received. `state` can be one of the following:

- `T2_MON_PRI_HDR`: a header (value names) must be printed
- `T2_MON_PRI_VAL`: the actual data must be printed
- `T2_MON_PRI_REPORT`: a report (similar to the plugin report) must be printed

A.5.10 `void pluginReport(FILE *stream)`

This function is used to report information regarding the plugin. This will appear in the final report.

A.5.11 `void onApplicationTerminate()`

This function is called once all the packets have been processed. Cleanup all used memory here.

A.5.12 `void saveState(FILE *stream)`

This function is used to save the state of the plugin. Tranalyzer can then restore the state in a future execution.

A.5.13 `void restoreState(char *str)`

This function is used to restore the state of the plugin. `str` represents the line written in saveState().

A.5.14 `void bufferToSink(outputBuffer_t *buffer)`

This callback is only required for sink (output) plugins.

A.6 Error, warning, and informational messages

Tranalyzer2 provides several macros to report errors, warnings, informations or simple messages:

`T2_PLOG()`	print a normal message (standard terminal colors)	`pluginName: message`
`T2_PINF()`	print an information message (blue)	`[INF] pluginName: message`
`T2_PWRN()`	print a warning message (yellow)	`[WRN] pluginName: message`
`T2_PERR()`	print an error message (red)	`[ERR] pluginName: message`

Note that `T2_PERR` always prints to `stderr`, while the other macros print to `stdout` or `PREFIX_log.txt` if Tranalyzer `-l` option was used.

Their usage is straightforward:

```
T2_PLOG("pluginName", "message %d", 42);
```

Note that a trailing newline is automatically added.

A.7 Generating Output

The following macros can be used to declare and append new columns to the output buffer. The `BV_APPEND_*` macros are used to declare a new column with a given `name`, description `desc` and type. The `OUTBUF_APPEND_*` macros are used to append a value `val` of the given type to the buffer `buf`.

`BV` **Macro**	**Type**	**Corresponding** `OUBUF` **Macro**
Unsigned values		
`BV_APPEND_U8(bv, name, desc)`	`bt_uint_8`	`OUTBUF_APPEND_U8(buf, val)`
`BV_APPEND_U16(bv, name, desc)`	`bt_uint_16`	`OUTBUF_APPEND_U16(buf, val)`
`BV_APPEND_U32(bv, name, desc)`	`bt_uint_32`	`OUTBUF_APPEND_U32(buf, val)`
`BV_APPEND_U64(bv, name, desc)`	`bt_uint_64`	`OUTBUF_APPEND_U64(buf, val)`
`BV_APPEND_H8(bv, name, desc)`	`bt_hex_8`	`OUTBUF_APPEND_H8(buf, val)`
`BV_APPEND_H16(bv, name, desc)`	`bt_hex_16`	`OUTBUF_APPEND_H16(buf, val)`
`BV_APPEND_H32(bv, name, desc)`	`bt_hex_32`	`OUTBUF_APPEND_H32(buf, val)`

`BV_APPEND_H64(bv, name, desc)`	`bt_hex_64`	`OUTBUF_APPEND_H64(buf, val)`

Signed values

`BV_APPEND_I8(bv, name, desc)`	`bt_int_8`	`OUTBUF_APPEND_I8(buf, val)`
`BV_APPEND_I16(bv, name, desc)`	`bt_int_16`	`OUTBUF_APPEND_I16(buf, val)`
`BV_APPEND_I32(bv, name, desc)`	`bt_int_32`	`OUTBUF_APPEND_I32(buf, val)`
`BV_APPEND_I64(bv, name, desc)`	`bt_int_64`	`OUTBUF_APPEND_I64(buf, val)`

Floating points values

`BV_APPEND_FLT(bv, name, desc)`	`bt_float`	`OUTBUF_APPEND_FLT(buf, val)`
`BV_APPEND_DBL(bv, name, desc)`	`bt_double`	`OUTBUF_APPEND_DBL(buf, val)`

String values

`BV_APPEND_STR(bv, name, desc)`	`bt_string`	`OUTBUF_APPEND_STR(buf, val)`
`BV_APPEND_STRC(bv, name, desc)`	`bt_string_class`	`OUTBUF_APPEND_STR(buf, val)`

Time values (timestamp and duration)[21]

`BV_APPEND_TIMESTAMP(bv, name, desc)`	`bt_timestamp`	`OUTBUF_APPEND_TIME(buf, sec, usec)`
`BV_APPEND_DURATION(bv, name, desc)`	`bt_duration`	`OUTBUF_APPEND_TIME(buf, sec, usec)`

IP values (network order)

`BV_APPEND_IP4(bv, name, desc)`	`bt_ip4_addr`	`OUTBUF_APPEND_IP4(buf, val)`
`BV_APPEND_IP6(bv, name, desc)`	`bt_ip6_addr`	`OUTBUF_APPEND_IP6(buf, val)`
`BV_APPEND_IPX(bv, name, desc)`	`bt_ipx_addr`	`OUTBUF_APPEND_IPX(buf, version, val)`[22]

If more flexibility is required the following macros can be used:

- `BV_APPEND(bv, name, desc, num_val, type1, type2, ...)`
- `OUTBUF_APPEND(buf, val, size)`

A.7.1 Repetitive Values

A repetitive value consists of a `uint32` representing the number of repetitions, followed by the actual repetitions.

All the `BV_APPEND` macros introduced in the previous section can be suffixed with `_R` to represent a repetitive value:

`BV_APPEND_U8(bv, name, desc)` (non-repetitive) ⇒ `BV_APPEND_U8_R(bv, name, desc)` (repetitive).

In addition, the following `OUTBUF` macros are available for repetitive values:

[21] Time values use an `uint64` for the seconds and an `uint32` for the micro-seconds

[22] Appends the IP `version` (`uint8`), followed by the IP. If `version` is 6, then calls `OUTBUF_APPEND_IP6(buf, val.IPv6.s6_addr[0]` else calls `OUTBUF_APPEND_IP4(buf, val.IPv4.s_addr`

OUTBUF Macro	Description	Type
OUTBUF_APPEND_OPTSTR(buf, val)	If val is NULL or empty, appends 0 (uint32) else appends 1 (uint32) and the string	bt_string, bt_string_class
OUTBUF_APPEND_NUMREP(buf, reps)	Appends the number of repetitions (uint32)[23]	

A.7.2 Column Names

Column names should be kept short and only contain characters in the following ranges: _, a-z, A-Z, 0-9. In addition, each "word" should start with an uppercase letter, e.g., myCol2. The '_' character should be used to name compound values, e.g., field1_field2. A good practice is to prefix each column name with the short name of the plugin, e.g., ftpDecode → ftpStat, ftpCNum

A.7.3 More Complex Output

Refer to Section A.7.

A.8 Accessible structures

Due to practical reasons all plugins are able to access every structure of the main program and the other plugins. This is indeed a security risk, but since Tranalyzer2 is a tool for practitioners and scientists in access limited environments the maximum possible freedom of the programmer is more important for us.

A.9 Important structures

A predominant structure in the main program is the flow table *flow* where the six tuple for the flow lookup timing information is stored as well as a pointer to a possible opposite flow. A plugin can access this structure by including the packetCapture.h header. For more information please refer to the header file.
Another important structure is the main output buffer mainOutputBuffer. This structure holds all standard output of activated plugins whenever a flow is terminated. The main output buffer is accessible if the plugin includes the header file main.h.

A.10 Generating output (advanced)

As mentioned in Section 2.12 there are two ways to generate output. The first is the case where a plugin just writes its arbitrary output into its own file, the second is writing flow-based information to a standard output file. We are now discussing the later case.
The standard output file generated by the Standard File sink plugin consists of a header, a delimiter and values. The header is generated using header information provided by each plugin, that writes output into the standard output file. During the initialization phase of the sniffing process, the core calls the printHeader() functions of these plugins. These functions return a single structure or a list of structures of type binary_value_t. Each structure represents a statistic. To provide a mechanism for hierarchical ordering, the statistic itself may contain one ore more values and one or more substructures. The structure contains the following fields:

[23] The correct number of values **MUST** then be appended.

Field name	Field type	Explanation
num_values	uint32_t	Amount of values in the statistic
subval	binary_subvalue_t*	Type definition of the values
name_value_short	char[128]	Short definition of the statistic
name_value_long	char[1024]	Long definition of the statistic
is_repeating	uint32_t	one, if the statistic is repeating, zero otherwise
next	binary_value_t*	used if the plugin provides more than one statistics

The substructure `binary_subvalue_t` is used to describe the values of the statistic. For each value, one substructure is required. For example, if `num_values` is two, two substructures have to be allocated. The substructures must be implemented as a continuous array consisting of the following fields:

Field name	Field type	Explanation
value_type	uint32_t	Type of the value
num_values	uint32_t	Amount of values in the statistic
subval	binary_subvalue_t*	Definition of the values
is_repeating	uint32_t	one, statistic is repeating, zero otherwise

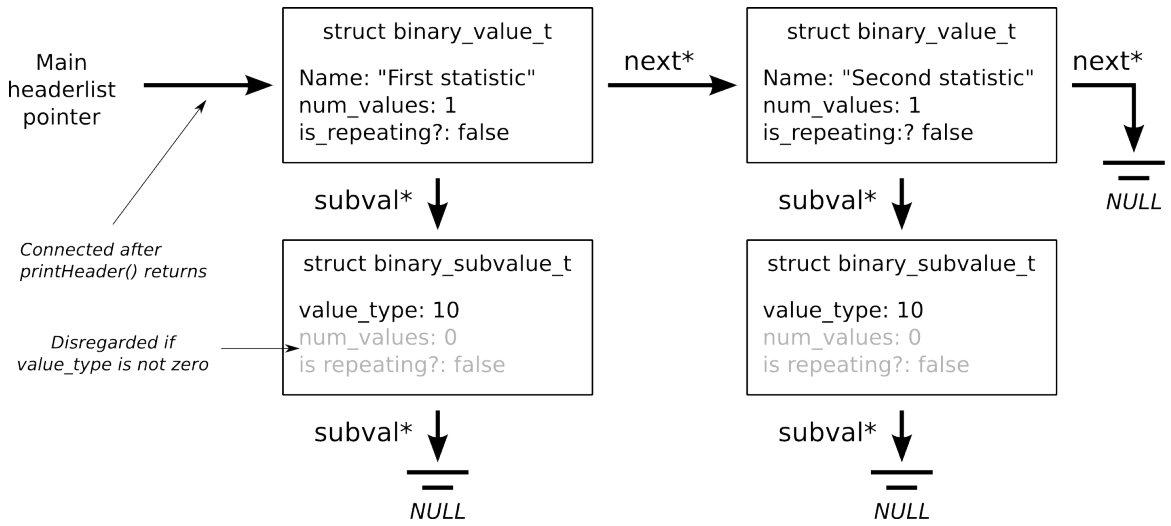
Compared to the `binary_value_t` representation two strings are omitted in the statistic's short and long description and the *next pointer but it contains a new field, the value type. Possible values for this new field are described in the enumeration `binary_types` defined in the header file `binaryValue.h`. If the field contains a value greater than zero the fields `num_values` and `subval` are ignored. They are needed if a subval contains itself subvalues. To indicate additional subvalues, the field `value_type` need to be set to zero. The mechanism is the same as for the `binary_value_t`.

The field `is_repeating` should be used if the number of values inside a statistic is variable; e.g. a statistic of a vector with variable length.

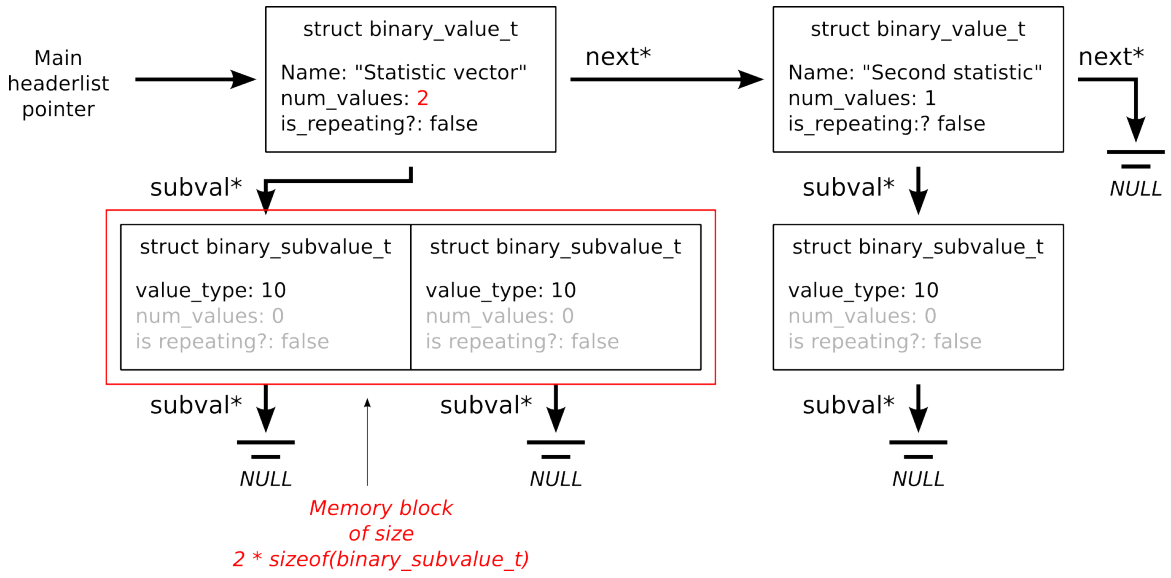
A.10.1 Examples

The following examples illustrate the usage of the said two structures:

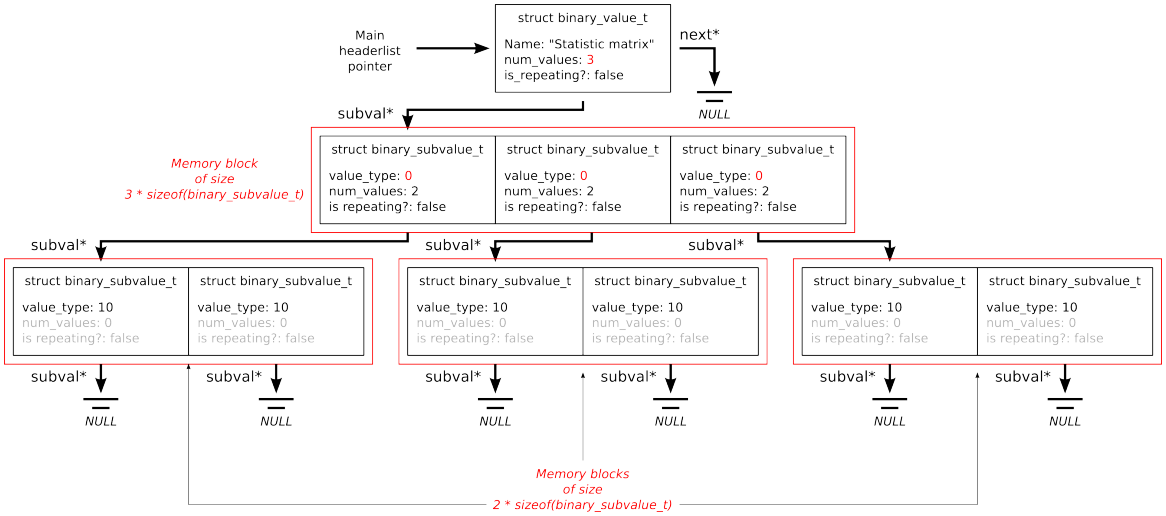
Example 1: Two Statistics each containing a single value If a plugin's output is consisting of two statistics each having a single value it needs to pass a list containing two structures of type `binary_value_t`. Both structures contain a substructure with the type of the single values. The following diagram shows the relationships between all four structures:



Example 2: A statistic composed of two values Now the output of the plugin is again two statistics, but the first statistic consists of two values; e.g. to describe a position on a grid. Therefore `num_values` is two and `subval*` points to a memory field of size two-times struct `binary_subvalue_t`. The subvalues themselves contain again the type of the statistic's values. Note: These values do not need to be identical.



Example 3: A statistic containing a complete matrix With the ability to define subvalues in subvalues it is possible to store multidimensional structures such as matrices. The following example illustrates the definition of a matrix of size three times two:



A.10.2 Helper functions

In order to avoid filling the structures by hand a small API is located in the header file `binaryValue.h` doing all the nitty-gritty work for the programmer. The therefore important four functions are described below.

`binary_value_t* bv_append_bv(binary_value_t* dest, binary_value_t* new)`
Appends a binary_value_t struct at the end of a list of binary_value_t structures and returns a pointer to the start of the list.

Arguments:

Type	Name	Explanation
binary_value_t*	dest	The pointer to the start of the list
binary_value_t*	new	The pointer to the new binary_value_t structure

binary_value_t* bv_new_bv (char* name_long, char* name_short, uint32_t is_repeating, uint32_t num_values...)

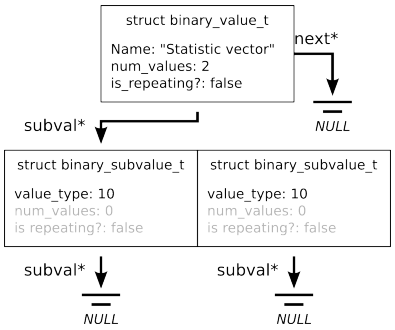
Generates a new structure of type binary_value_t and returns a pointer to it

Arguments:

Type	Name	Explanation
char*	name_long	a long name for the statistic
char*	name_short	a short name for the statistic
uint32_t	is_repeating	one, if the statistic is repeating, zero otherwise
uint32_t	num_values	the number of values for the statistic
int	...	the types of the statistical values, repeated `num_values`-times

The function creates a `binary_value_t` structure and sets the values. In addition, it creates an array field with num_values binary_subvalue_t structures and fills the value types provided in the variable argument list.

Example: The call `bv_new_bv("Statistic vector", "stat_vec", 2, 0, bt_uint_64, bt_uint_64)` creates the following structures:




```
binary_value_t* bv_add_sv_to_bv (binary_value_t* dest, uint32_t pos,
                                 uint32_t is_repeating, uint32_t num_values, ...)
```

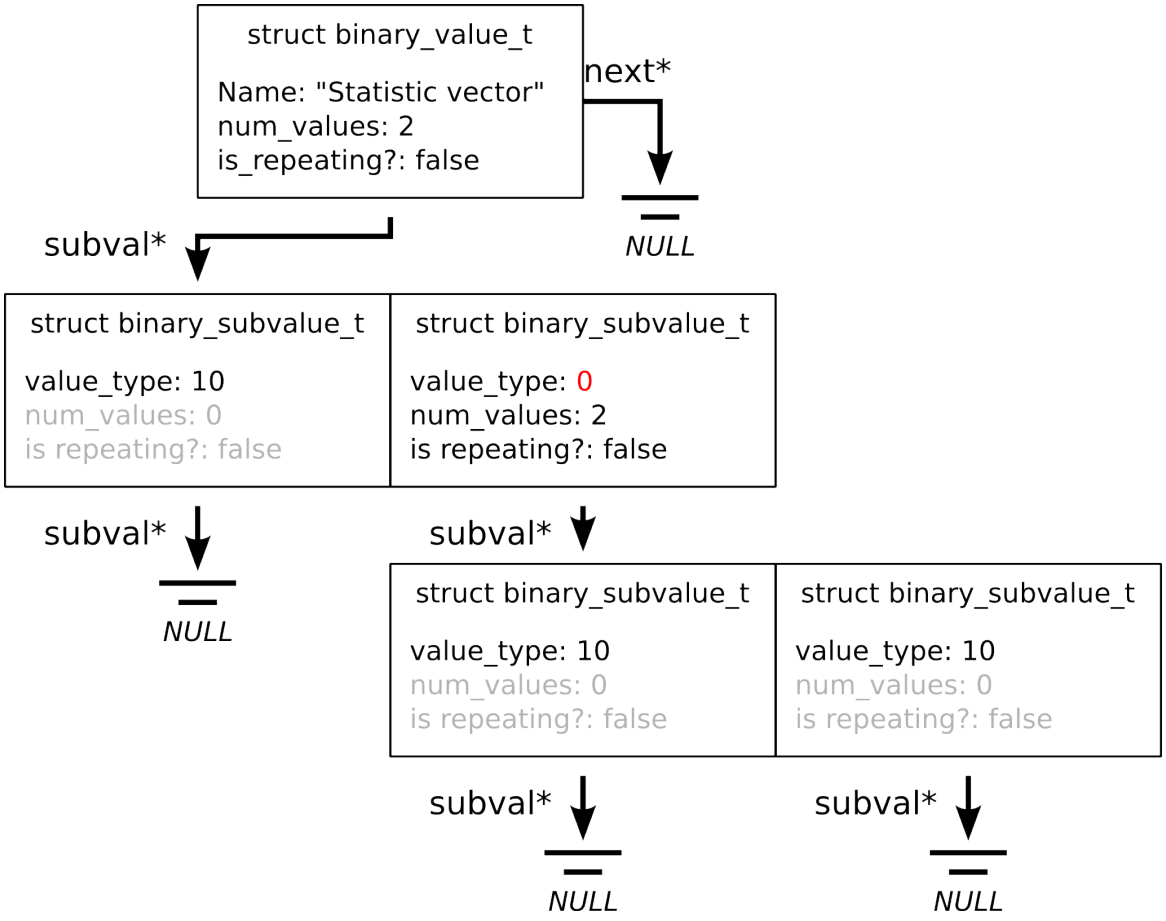
Replaces a subvalue in a `binary_value_t` *structure with a new substructure that contains additional substructures and returns a pointer to the parent binary value.*

Arguments:

Type	Name	Explanation
binary_value_t*	dest	the pointer to the parent binary value
uint32_t	pos	the position of the substructure to be replaced, starting at 0
uint32_t	is_repeating	one, if the subvalue is repeating, zero otherwise
uint32_t	num_values	the number of values in the subvalue
int	...	the types of the statistical values, repeated `num_values`-times

This function is only valid if `dest` is already a complete statistic containing all necessary structures.

Example: Let *dest* be a pointer to the `binary_value_t` structure from the example above. A call to the function `bv_add_sv_to_bv(dest, 1, 0, 2, bt_uint_64, bt_uint_64)` replaces the second substructure with a new substructure containing two more substructures:




```
binary_value_t* bv_add_sv_to_sv (binary_subvalue_t* dest, uint32_t pos,
                                 uint32_t is_repeating, uint32_t num_values, ...)
```

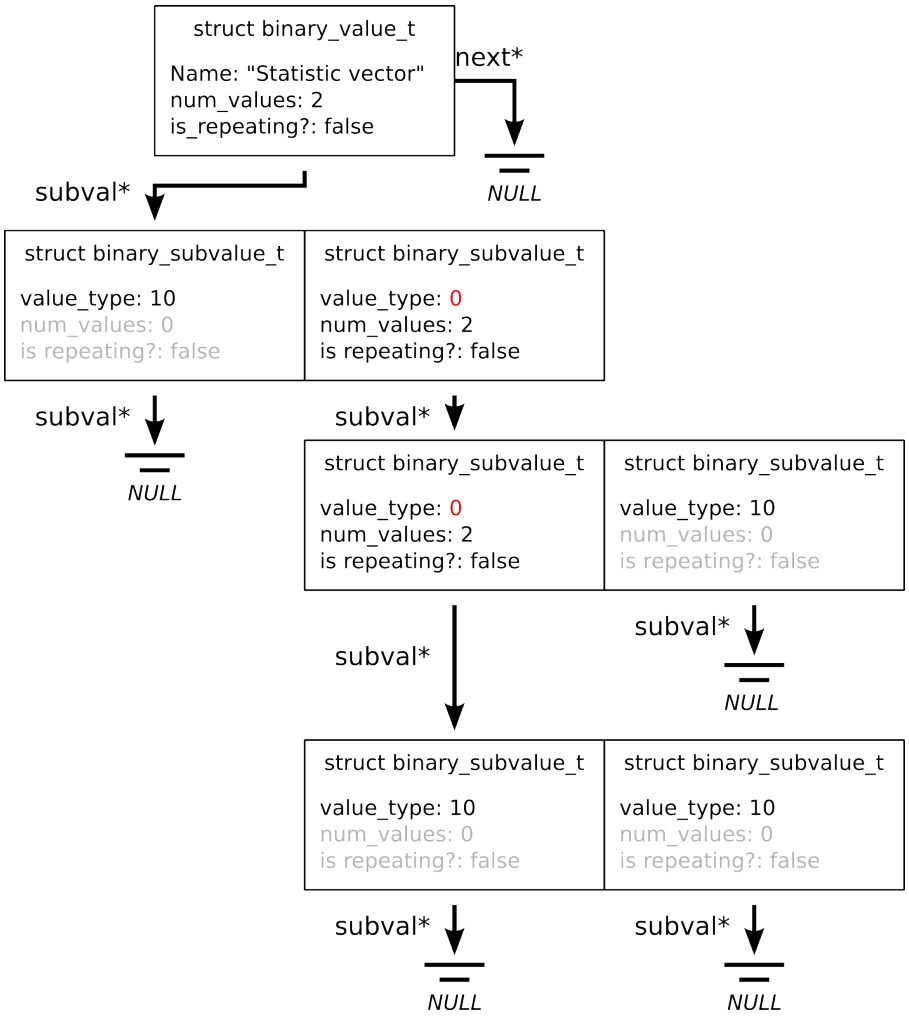
Replaces a subvalue in a `binary_subvalue_t` *structure with a new substructure that contains additional substructures and returns a pointer to the parent binary subvalue.*

Arguments:

Type	Name	Explanation
binary_subvalue_t*	dest	Pointer to the parent binary subvalue
uint32_t	pos	Position of the substructure to be replaced, starting at 0
uint32_t	is_repeating	one, if the subvalue is repeating, zero otherwise
uint32_t	num_values	Number of values in the subvalue
int	...	Types of the statistical values, repeated *num_values*-times

For all hierarchical deeper located structures than above the function described above is required.

Example: Let *dest* be a pointer to the subvalue structure being replaced in the example above. A call to the function `bv_add_sv_to_sv(dest, 0, 0, 2, bt_uint_64, bt_uint_64)` replaces *dest's* first the substructure with a new substructure containing two more substructures:



A.10.3 Writing into the standard output

Standard output is generated using a buffer structure. Upon the event `onFlowTerminate` (see A.14.6) Plugins write all output into this buffer. It is strongly recommended using the function `outputBuffer_append(outputBuffer_t* buffer, char* output, size_t size_of_output)`.
Arguments:

Type	Name	Explanation
outputBuffer_t*	buffer	the pointer to the standard output buffer structure, for standard

Type	Name	Explanation
		output, this is `main_output_buffer`
char*	output	a pointer to the output, currently of type char
size_t	size_of_output	the length of field *output* in single bytes

The output buffer is send to the *output sinks* after all plugins have stored their information.

Example: If a plugin wants to write two statistics each with a single value of type `uint64_t` it first has to commit its `binary_value_t` structure(s) (see section above). During the call of its `onFlowTerminate()` function the plugin writes both statistical values using the append function:

```
outputBuffer_append(main_output_buffer, (char*) &value1, 4);
outputBuffer_append(main_output_buffer, (char*) &value2, 4);
```

Where `value1` and `value2` are two pointers to the statistical values.

A.11 Writing repeated output

If a statistic could be repeated (field `is_repeating` is one) the plugin has first to store the number of values as `uint32_t` value into the buffer. Afterwards, it appends the values.

Example: A plugin's output is a vector of variable length, the values are of type `uint16_t`. For the current flow, that is terminated in the function `onFlowTerminate()`, there are three values to write. The plugin first writes a field of type `uint32_t` with value three into the buffer, using the append function:

```
outputbuffer_append(main_output_buffer, (char*) &numOfValues, sizeof(uint32_t));
```

Afterwards, it writes the tree values.

A.12 Important notes

- IP addresses (bt_ip4_addr or bt_ip6_addr) or MAC addresses (bt_mac_addr) are stored in network order.
- Strings are of variable length and need to be stored with a trailing zero bit ('\0').

A.13 Administrative functions

Every plugin has to provide five administrative functions. The first four are mandatory while the last one is optional. For convenience, the following two macros can be used instead:

- `T2_PLUGIN_INIT(name, version, t2_v_major, t2_v_minor)`
- `T2_PLUGIN_INIT_WITH_DEPS(name, version, t2_v_major, t2_v_minor, deps)`

For example, to initialize `myPlugin`:

```
T2_PLUGIN_INIT_WITH_DEPS("myPlugin", "0.8.3", 0, 8, "tcpFlags,basicStats")
```

Function name	Return type	Explanation
get_plugin_name()	char*	a unique name of the plugin, not necessarily the filename. All characters except the comma is allowed.
get_plugin_version()	char*	a version number, usually a dot separated 3 tuple (x.y.z)
get_supported_tranalyzer_version_major()	unsigned int	The minimum major version number of the main program being supported by the plugin
get_supported_tranalyzer_version_minor()	unsigned int	The minimum minor version number in combination with the minimum major version number of the main program being supported by the plugin
get_dependencies()	char*	if exists, the plugin loader checks the availability of the plugin names returned by this function. The plugin names have to be separated by a comma. White spaces, tabs or any other characters are **not** treated as name separators.

The existence of these functions is checked during the plugin initialization phase one and two, as highlighted in Figure 6.

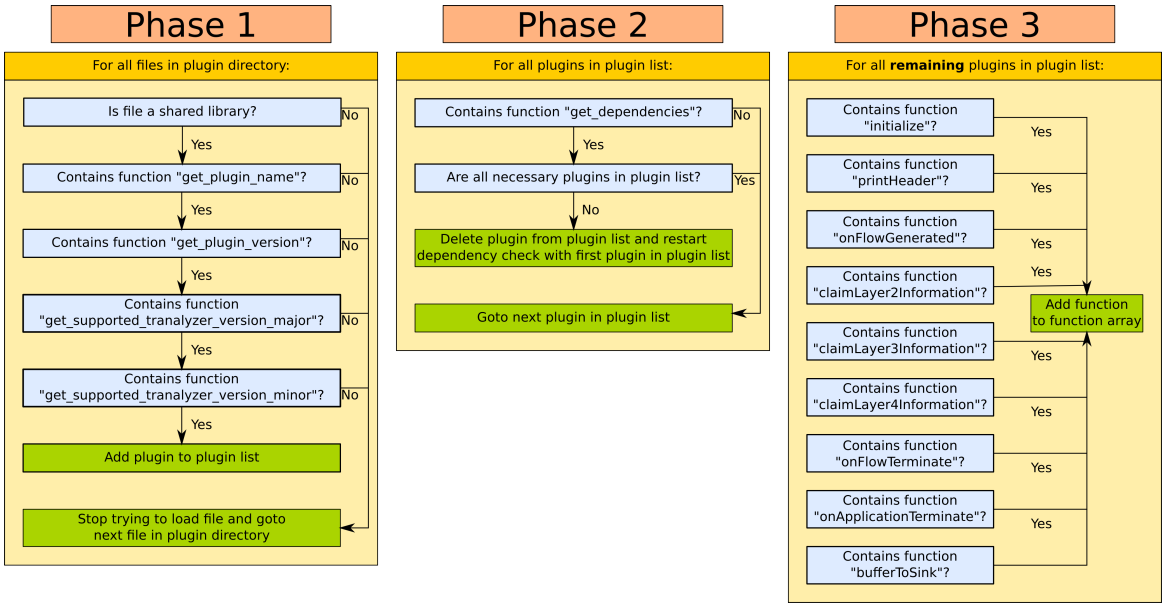


Figure 6: *Processing of the plugin loading mechanism*

A.14 Processing functions

During flow analysis Tranalyzer2 generates several *events* based on the status of the program, the inspected OSI layer of the current packet or the status of the current flow. These events consist of specific function calls provided by the plugins. The implementation of the event functions is dependent on the required action of a plugin to be carried out upon a certain event.

A.14.1 Event: initialize()

Event / function name	Return type	Parameters
initialize	void	—

The `initialize` event is generated before the program activates the packet capturing phase. After Tranalyzer2 has initialized its internal structures it grants the same phase to the plugins. Therefore temporary values should be allocated during that event by using a C `malloc`.

A.14.2 Event: printHeader()

Event / function name	Return type	Parameters
printHeader	binary_value_t*	—

This event is also generated during the initialization phase. With this event the plugin providing data to the standard output file signals the core what type of output they want to write (see A.7). The function returns a pointer to the generated `binary_value_t` structure or to the start pointer of a list of generated `binary_value_t` structures.

A.14.3 Event: onFlowGenerated()

Event / function name	Return type	Parameters
onFlowGenerated	void	packet_t *packet, unsigned long flowIndex

This event is generated every time Tranalyzer2 recognizes a new flow not present in the flow table. The first parameter is the currently processed packet, the second denotes the new generated flow index. As long as the flow is not terminated the flow index is valid. After flow termination the flow number is reintegrated into a list for later reuse.

A.14.4 Event: claimLayer2Information()

Event / function name	Return type	Parameters
claimLayer2Information	void	packet_t *packet

This event is generated for every new packet comprising of a valid and supported layer two header, e.g. Ethernet as default. This is the first event generated after libpcap dispatches a packet and before a lookup in the flow table happened.

At this very point in time no tests are conducted for higher layer headers. If a plugin tries to access higher layer structures it has to test itself if they are present or not. Otherwise, at non-presence of higher layers an unchecked access can result in a `NULL` pointer access and therefore in a possible segmentation fault! We recommend using the subsequent two events to access higher layers.

A.14.5 Event: claimLayer4Information()

Event / function name	Return type	Parameters
claimLayer4Information	void	packet_t *packet, unsigned long flowIndex

This event is generated for every new packet containing a valid and supported layer four header. The current supported layer four headers are TCP, UDP and ICMP. This event is called after Tranalyzer2 performs a lookup in its flow table and eventually generates an `onFlowGenerated` event. Implementation of other protocols such as IPsec or OSPF are planned.

A.14.6 Event: onFlowTerminate()

Event / function name	Return type	Parameters
onFlowTerminate	void	unsigned long flowIndex

This event is generated every time Tranalyzer2 removes a flow from its active status either due to timeout or protocol normal or abnormal termination. Only during this event, the plugins write output to the standard output.

A.14.7 Event: onApplicationTerminate()

Event / function name	Return type	Parameters
onFlowTerminate	void	—

This event is generated shortly before the program is terminated. At this time no more packets or flows are processed. This event enables the plugins to do memory housekeeping, stream buffer flushing or printing of final statistics.

A.14.8 Event: bufferToSink()

Event / function name	Return type	Parameters
bufferToSink	void	outputBuffer* buffer

The Tranalyzer core generates this event immediately after the `onFlowTerminate` event with the main output buffer as parameter. A plugin listening to this event is able to write this buffer to a data sink. For example the binSink plugin pushes the output into the `PREFIX_flows.bin` file.

A.15 Timeout handlers

A flow is terminated after a certain timeout being defined by so called *timeout handlers*. The default timeout value for a flow is 182 seconds. The plugins are able to access and change this value. For example, the tcpStates plugin changes the value according to different connection states of a TCP flow.

A.15.1 Registering a new timeout handler

To register a new timeout handler, a plugin has to call the `timeout_handler_add(float timeout_in_sec)` function. The argument is the new timeout value in seconds. Now the plugin is authorized by the core to change the timeout of a flow to the registered timeout value. Without registering a timeout handler the test is unreliable.

A.15.2 Programming convention and hints

- A call of `timeout_handler_add` should only happen during the initialization function of the plugin.
- Registering the same timeout value twice is no factor.
- Registering timeout values in fractions of seconds is possible, see tcpStates plugin.

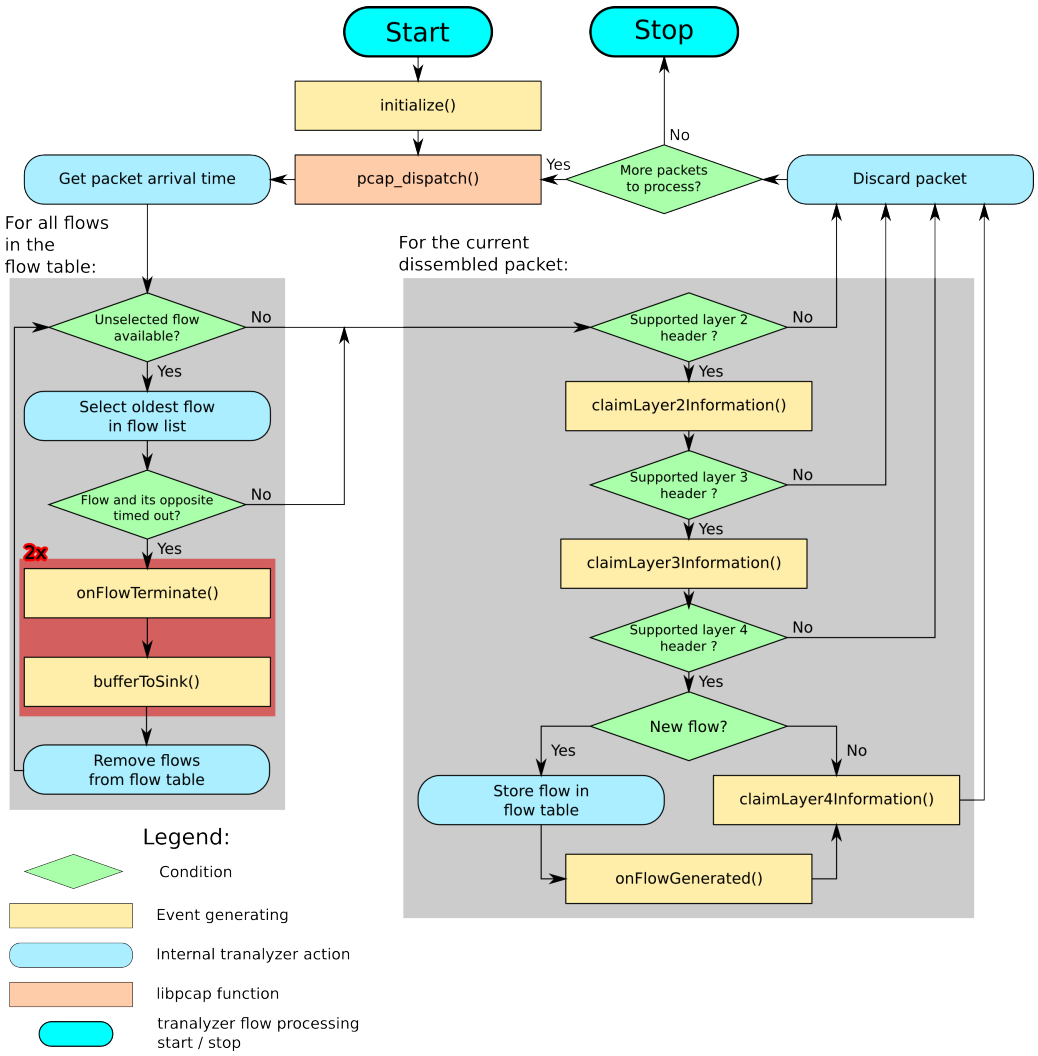


Figure 7: *Tranalyzer packet processing and event generation.*

B Importing Tranalyzer Flows in Splunk

B.1 Prerequisites

- Tranalyzer version 0.6.x is installed with standard/default plugins,
- Splunk 6.5.x is installed, Splunk account exists,
- At least one network interface (Ethernet or WLAN) has network traffic.

B.2 Select Network Interface

Determine the network interface name by entering the following command:

```
ifconfig
```

at the terminal command line. In the output look for the interface name which has the IP address where the network traffic should be collected from:

```
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST>
mtu 1500 inet 10.20.6.79 netmask 0xfffffc00 broadcast 10.20.7.255
```

B.3 Configure Tranalyzer jsonSink Plugin

Go to *tranalyzer2-0.6.XlmY/trunk/jsonSink/src/jsonSink.h* and set the configuration parameters as needed:

```
#define SOCKET_ON             1 // Whether to output to a socket (1) or file (0)
#define SOCKET_ADDR "127.0.0.1" // address of the socket
#define SOCKET_PORT        5000 // port of the socket
```

Set `SOCKET_ON` to `1` to configure the output to a socket. Set the IP address of the destination server which should receive the data stream. If the localhost will be the destination, leave the default setting `"127.0.0.1"`. Set the socket server port of the destination.

B.4 Recompile the jsonSink Plugin

Enter the following command:

```
tranalyzer2-0.6.8lm4/trunk/jsonSink/autogen.sh
```

Make sure that the plugin is compiled successfully. In this case the following message will be shown at the command line:

```
Plugin jsonSink copied into USER_DIRECTORY/.tranalyzer/plugins
```

B.5 Start Tranalyzer2

Start generating flow records by launching Tranalyzer2 with the interface name determined on the previous step and setting a file name as the command line arguments by entering the command:

```
tranalyzer -i en0 -w test1 &
```

Note that the file name is optional for JSON stream import, if file name is not indicated the records will be shown in the standard output (besides being streamed over the configured TCP socket).

B.5.1 Check File Output

Check that the flow records are written to the file by entering the command:

```
tail -f test1_flows.txt
```

Flow records should be shown in the terminal.

B.5.2 Collect Traffic

Let Tranalyzer2 run and collect network traffic.

B.6 Start Splunk

Start Splunk by entering the following command:

```
splunk start
```

in the directory where Splunk is installed. Wait for the confirmation message that Splunk is up and running:

```
The Splunk web interface is at http://splunk_hostname:8000
```

B.7 Login to Splunk, Import and Search Data

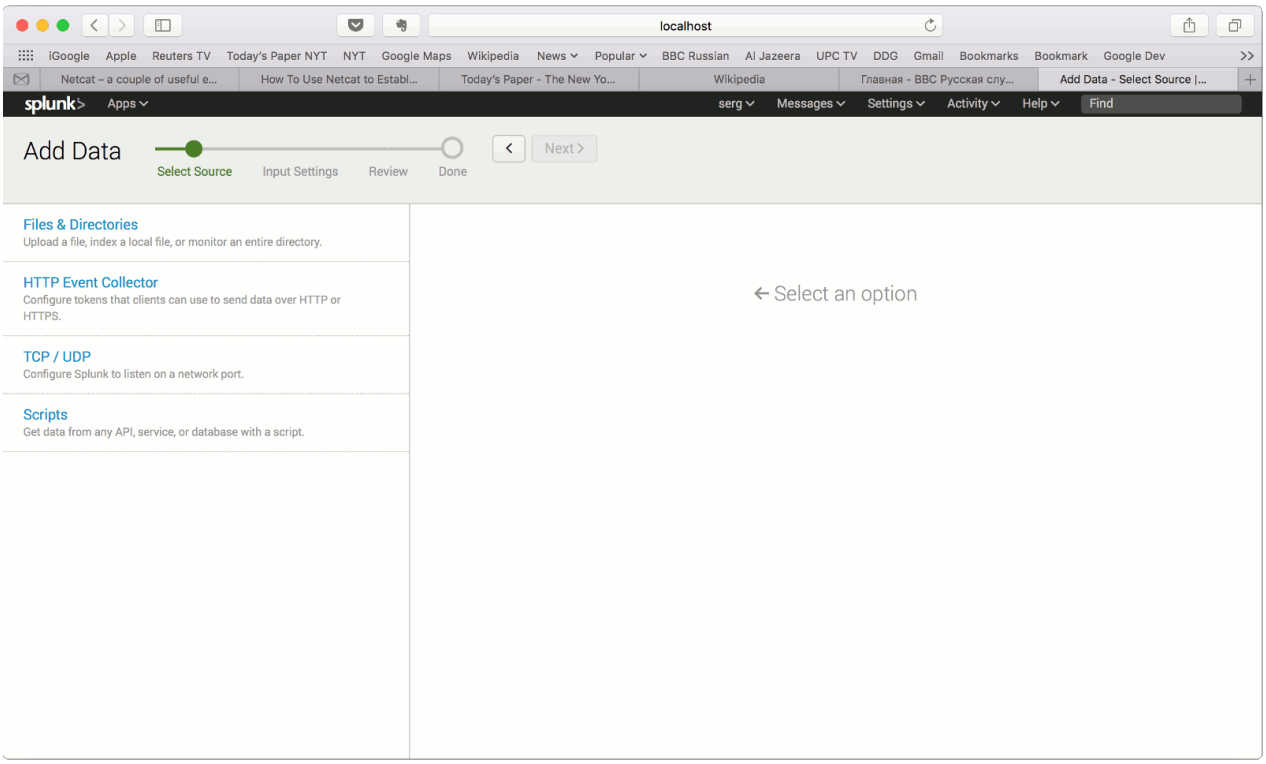


Figure 8: *Select "Add Data".*

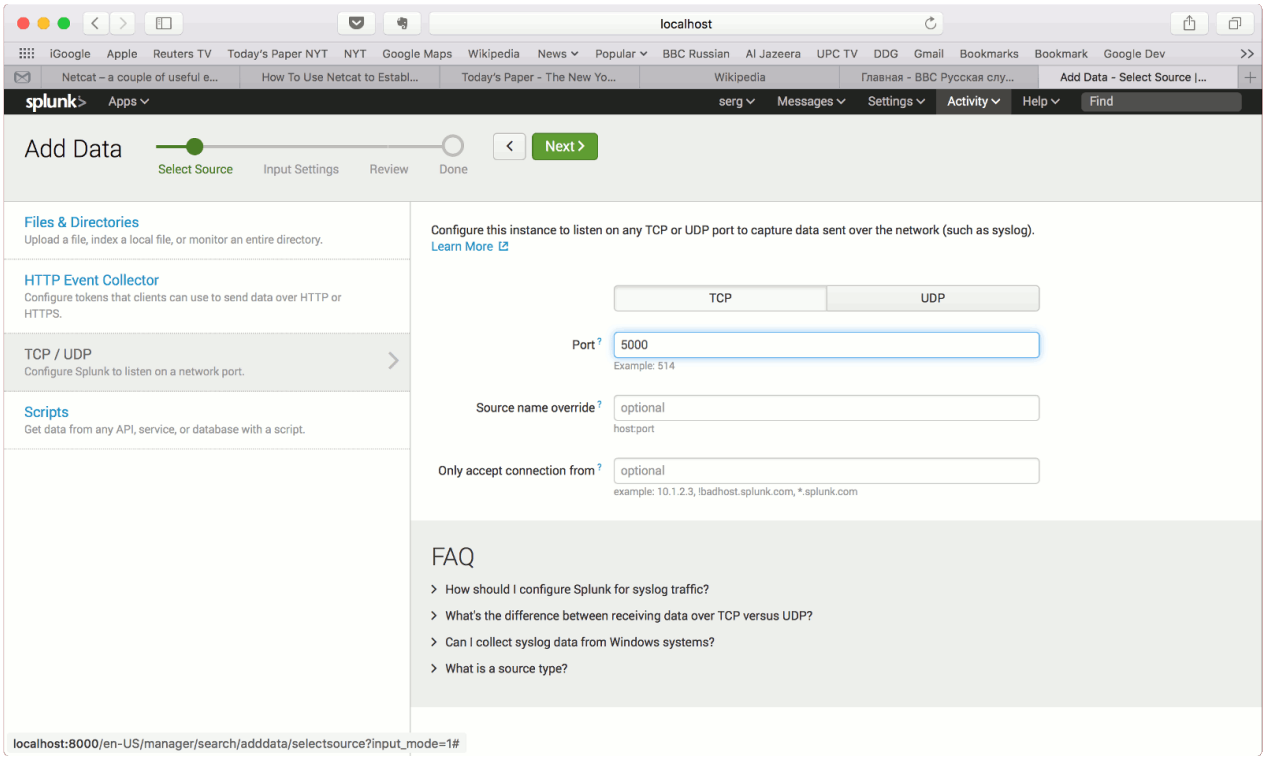


Figure 9: *Select "TCP/UDP" and set protocol to "TCP" and set the correct port number (same as in the Tranalyzer2 plugin configuration file, in this example — 5000).*

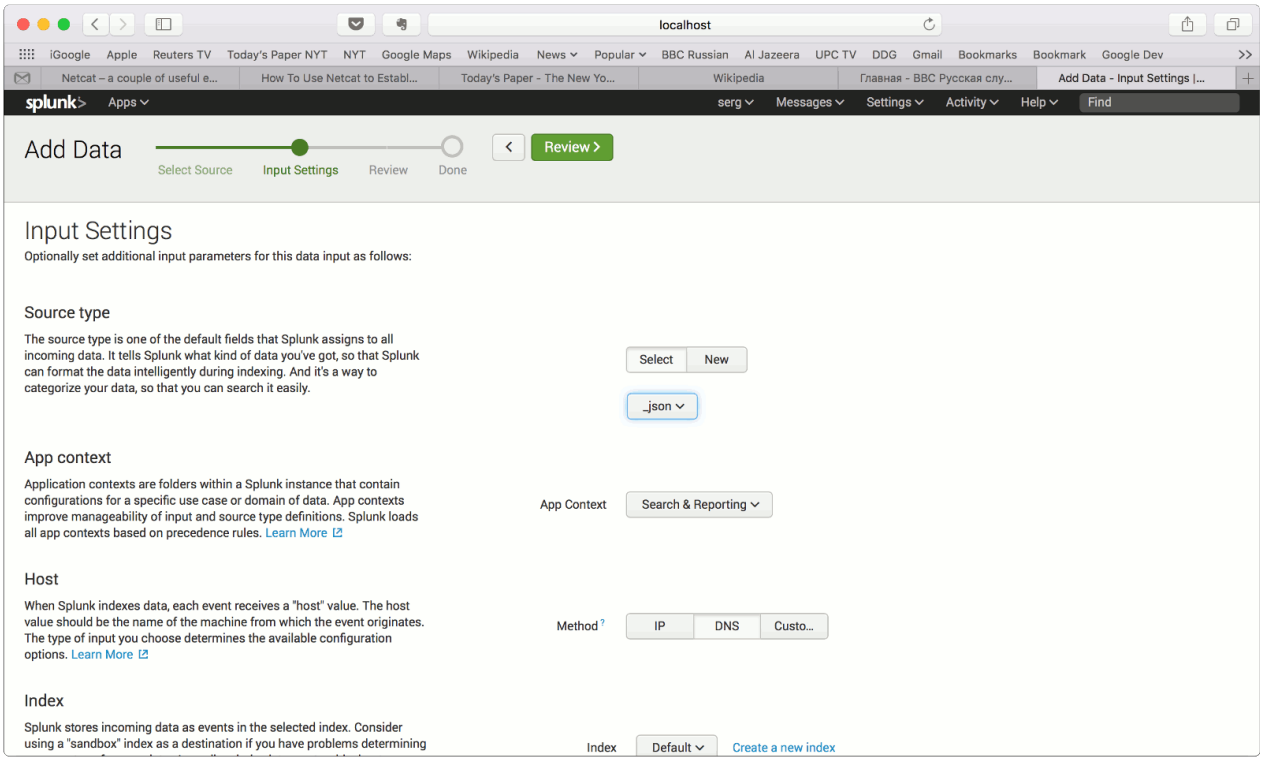


Figure 10: *Select "_json" as Source Type and proceed to "Review".*

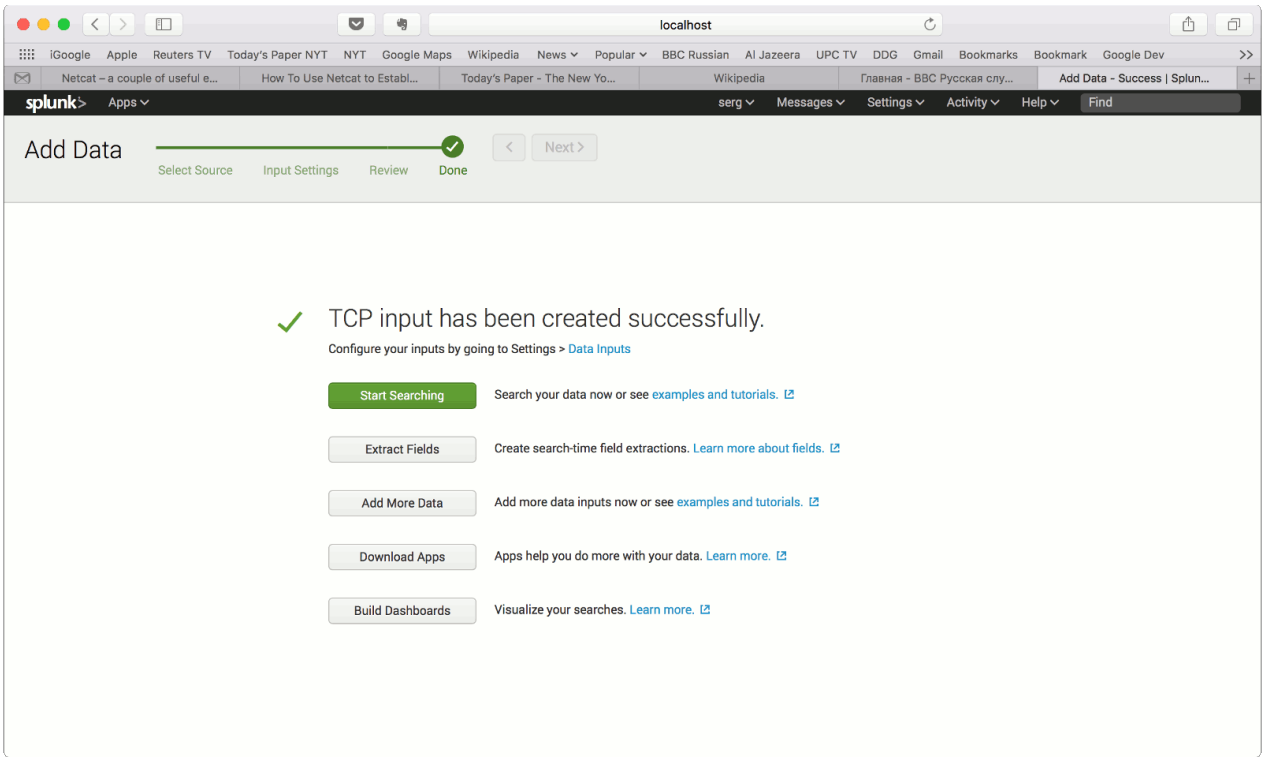


Figure 11: *Select "Start Searching" to make sure that the data is being received by Splunk.*

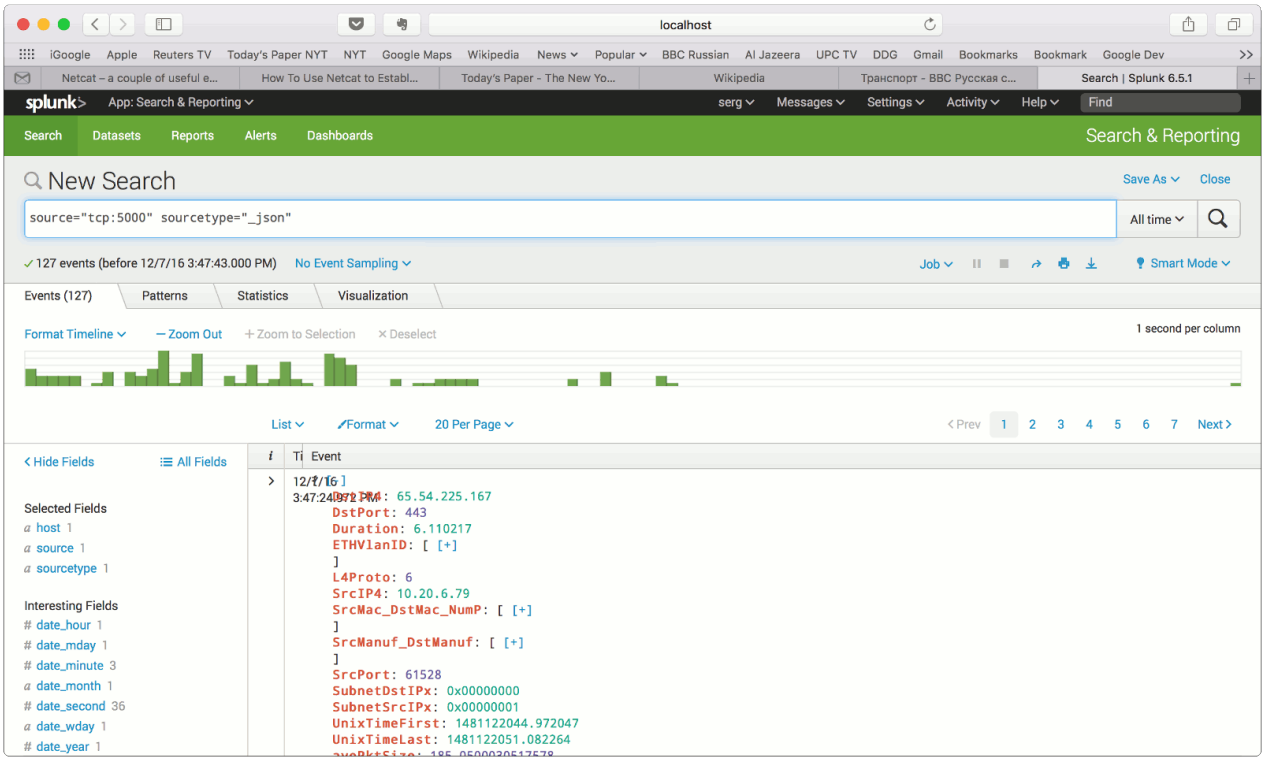


Figure 12: *Note that the data is being received, but the Tranalyzer2 specific data record field are not shown yet.*

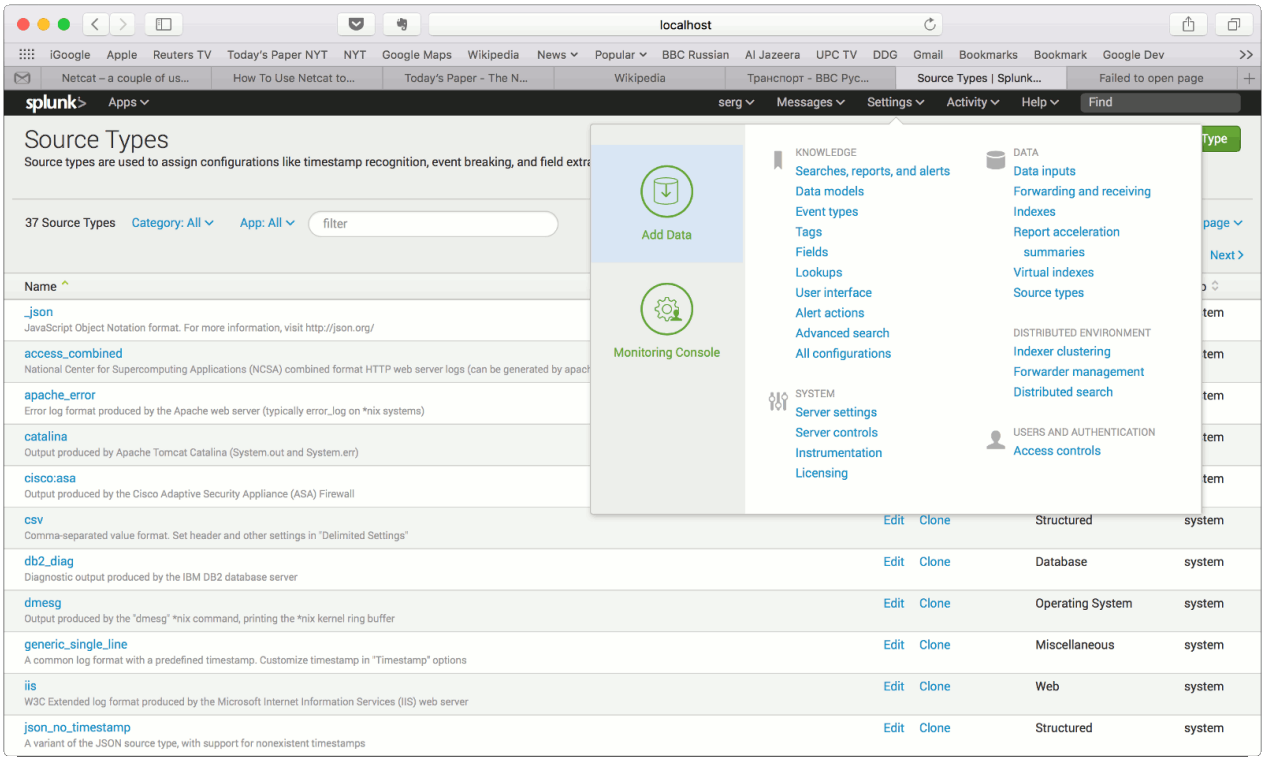


Figure 13: *Go to "Settings"->"DATA"->"Source Types" and click on "_json" data source type to edit it.*

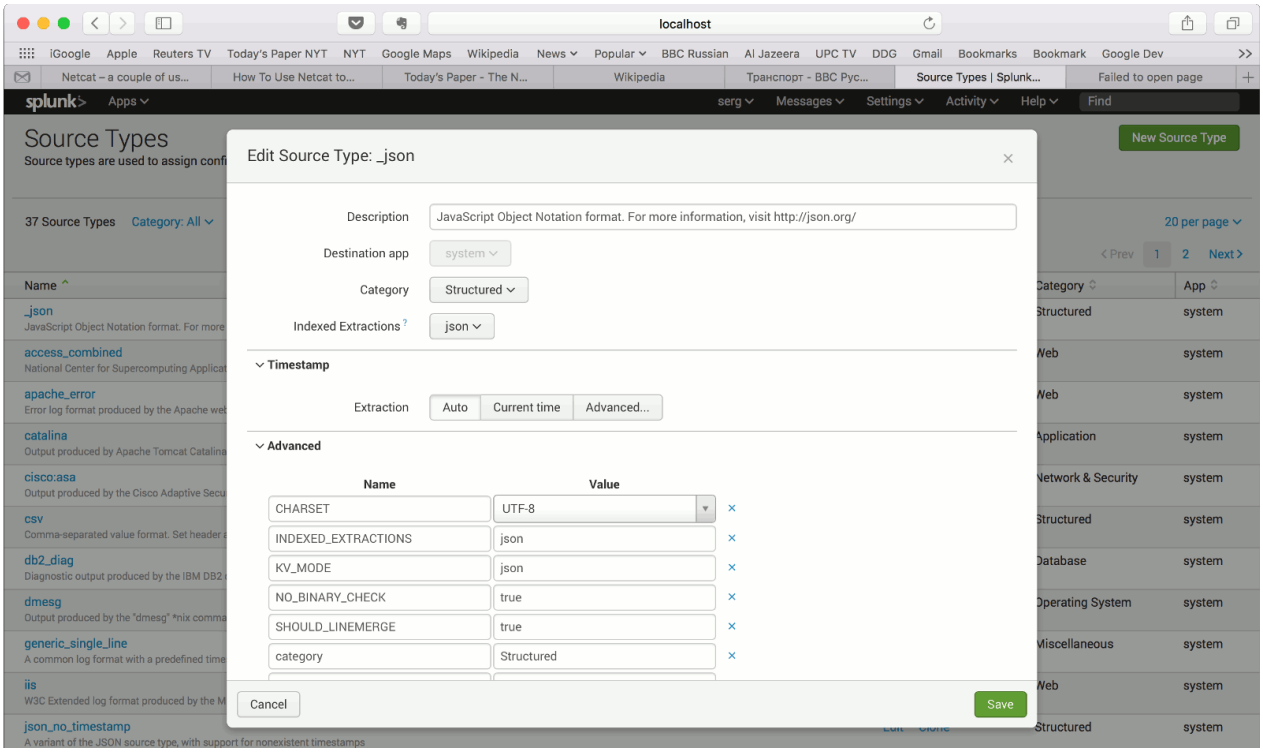


Figure 14: *Change option "KV_MODE" from "none" to "json" and save the changes.*

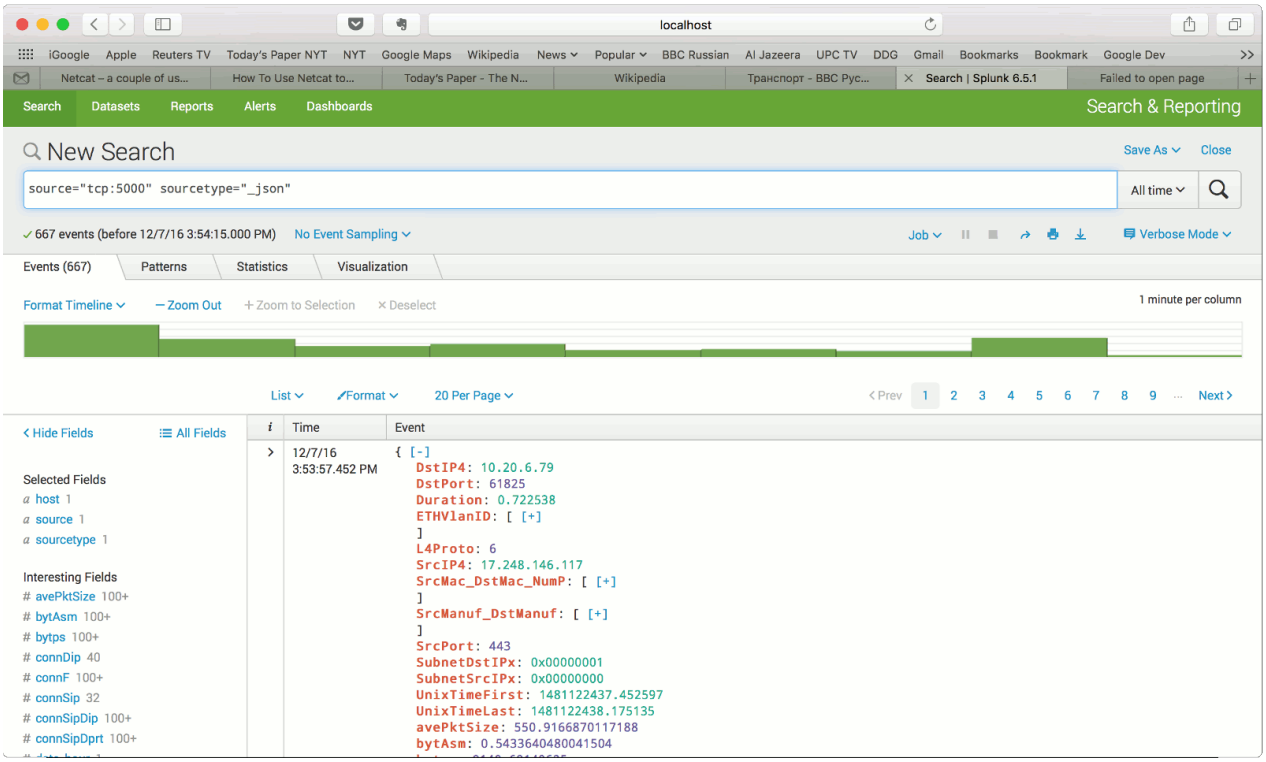


Figure 15: *Return to the Search window and make sure that the Tranalyzer2 specific fields are recognized by Splunk.*

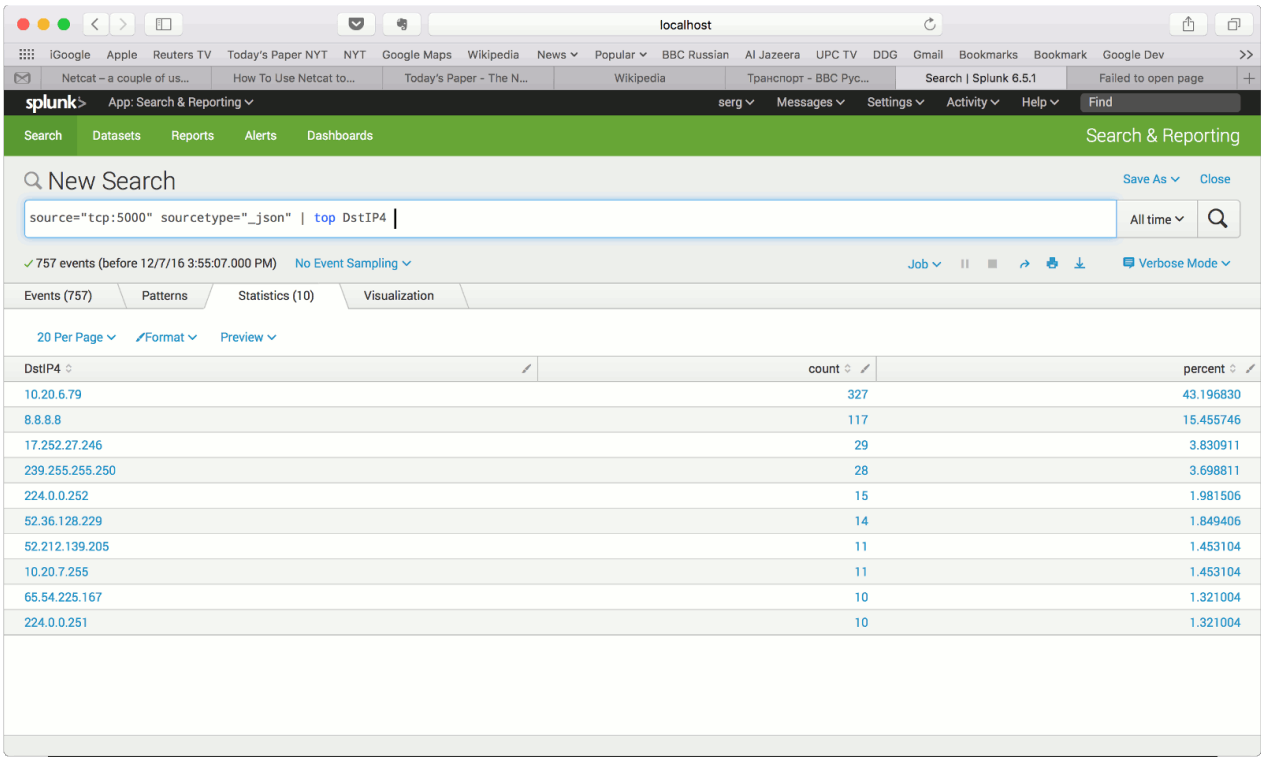


Figure 16: *Query data, e.g. show top destination IP addresses by number of the records.*

C Advanced Performance Enhancements with PF_RING

Under certain circumstances, e.g., large quantities of small packets, the kernel might drop packets. This happens due to the normal kernel dispatching which is known to be inefficient for packet capture operations. The capturing process can be devised more efficiently by changing the kernel as in packet_mmap, but then a patched libpcap is required which is not available yet.[24] Another option is pf_ring. Its kernel module passes the incoming packets in a different way to the user process.[25]

Requirements

- Kernel version prior to 3.10. [26]
- All packages needed for building a kernel module, names are distribution-dependent
- A network interface which supports NAPI polling by its driver.
- optional: A network card which supports Direct Network Interface Card (NIC) access (DNA).[27]

Quick setup

Download PF_RING from a stable tar ball or development source at http://www.ntop.org/get-started/download/. In order to build the code the following commands have to executed in a bash window:

```
cd PF_RING/kernel
make && sudo make install
modprobe pf_ring
```

Figure 17: *building kernel module*

Tranalyzer2 requires at least libpfring and libpcap-ring which can be installed the following way:

```
cd PF_RING/userland
cd lib
make && sudo make install
cd ..
cd libpcap
make && sudo make install
```

Figure 18: *basic userland*

You may like to install other tools such as tcpdump. Just install it the same way as described above.
NOTE: The *pf_ring.ko* is loaded having the `transparent_mode=0` by default which enables NAPI polling. If you use a card with special driver support for DNA you may want to compile the driver and load *pf_ring.ko* in a different mode.[28]

[24] See https://www.kernel.org/doc/Documentation/networking/packet_mmap.txt for more information
[25] See http://www.ntop.org/products/pf_ring/
[26] Presently when composing this document there is no patch for the depreciation of `create_proc_read_entry()` function. See: https://lkml.org/lkml/2013/4/11/215
[27] documentation: http://www.ntop.org/products/pf_ring/DNA/
[28] See: `man modprobe.d`

Load on boot

Since this seems to be difficult for many users the load procedure is described in the following.
Depending on your distribution or to be more specific, the init system your distribution uses at boot time may be somewhere different. In systemd [29] create a file with a '.conf' ending at */etc/modules-load.d/* which contains just the text `pf_ring`, the module name without the '.ko' ending.[30]
Ubuntu uses */etc/modules* as a single file where you can add a line with the module name.[31]

```
systemd
echo pf_ring > /etc/modules-load.d/pfring.conf
OR
ubuntu
echo pf_ring >> /etc/modules
```

Figure 19: *on-boot kernel module load examples*

New kernel

Once in a while there is indeed a new kernel available. If you want to use pf_ring afterwards do not forget to recompile the kernel module, or set up `dkms`.

[29] More info: http://www.freedesktop.org/wiki/Software/systemd/
[30] For more info: `man modules-load.d`
[31] See: `man modules`

D Status

This section summarises the available plugins. For each plugin, a brief description is provided, along with the development status (pre-alpha, alpha, beta, release-candidate, release or deprecated).

D.1 Global Plugins

Plugin Name	Number	Description	Status
protoStats	001	Overall statistics about protocols	release

D.2 Basic Plugins

Plugin Name	Number	Description	Status
basicFlow	100	Overall statistics plugin	release
macRecorder	110	MAC addresses and manufacturers	release
portClassifier	111	Classification based on port numbers	release
basicStats	120	Basic statistics	release
connStat	500	Connection statistics	release

D.3 L2 Protocol Plugins

Plugin Name	Number	Description	Status
arpDecode	200	ARP	beta
cdpDecode	207	CDP	beta
lldpDecode	206	LLDP	beta
stpDecode	203	STP	alpha

D.4 Protocol Plugins

Plugin Name	Number	Description	Status
dhcpDecode	250	DHCP	release
dnsDecode	251	DNS	beta
ftpDecode	301	FTP	release
httpSniffer	310	HTTP	release
icmpDecode	140	ICMP	release
igmpDecode	204	IGMP	alpha
ircDecode	401	IRC	beta
modbus	450	Modbus	beta
ntpDecode	205	NTP	release
ospfDecode	202	OSPF	release
popDecode	304	POP	release

Plugin Name	Number	Description	Status
radiusDecode	255	RADIUS	beta
sctpDecode	135	SCTP	beta
smbDecode	385	SMB	beta
smtpDecode	303	SMTP	release
snmpDecode	386	SNMP	beta
sshDecode	309	SSH	beta
sslDecode	311	SSL/TLS, OpenVPN	release
stunDecode	601	STUN, TURN, NAT-PMP	beta
syslogDecode	260	Syslog	release
tcpFlags	130	TCP flags	release
tcpStates	132	TCP states	release
telnetDecode	305	Telnet	release
tftpDecode	300	TFTP	release
voipDetector	410	VoIP	release
vrrpDecode	220	VRRP	beta

D.5 Application Plugins

Plugin Name	Number	Description	Status
pwX	602	Password extractor	release
regex_pcre	605	PCRE	release

D.6 Math Plugins

Plugin Name	Number	Description	Status
descriptiveStats	702	Descriptive statistics	release
entropy	710	Entropy	beta
nFrstPkts	700	Statistics over the first *N* packets	release
pktSIATHisto	701	Histograms of packet size and inter-arrival times	release
wavelet	720	Wavelet	beta

D.7 Classifier Plugins

Plugin Name	Number	Description	Status
fnameLabel	899	Classification based on filename	beta
nDPI	112	Classification based on content analysis	release
geoip	116	Classification based on content analysis	release
p0f	779	OS Classification based on content analysis (SSL)	release
tp0f	117	OS Classification based on content analysis	release

D.8 Output Plugins

Plugin Name	Number	Description	Status
binSink	900	Binary output into a flow file	release
findexer	961	Produces a binary index mapping flow index and packets	release
jsonSink	903	Produces a JSON file	release
mongoSink	926	Output into a MongoDB database	beta
mysqlSink	925	Output into a MySQL database	beta
netflowSink	904	Netflow output format for existing Cisco tools	beta
pcapd	960	Stores packets from specific flows in pcap files	release
psqlSink	923	Output into a PostgreSQL database	beta
socketSink	910	Binary output into a TCP/UDP socket	release
sqliteSink	924	Output into a SQLite database	beta
txtSink	901	Text output into a flow file	release

E TODO

This section lists some features, capabilities and plugins which Tranalyzer is currently missing. Feel free to pick a task or two and contribute code, plugins or ideas.

E.1 Features

- Anonymisation
- Endianness independence
- Support for NetMon dump files
- Stream reassembly and reordering

E.2 Plugins

- SS7
- IMAP
- NFS, iSCSI
- XMPP
- Routing (EIGRP, RIPv2, HSRP (Cisco), . . .)
- Chat (MSN, IRC, YMSG, . . .)
- Torrent, Gnutella
- TDS (Tabular Data Stream, Microsoft)
- DCE/RPC
- LDAP
- X11
- Dropbox
- OCSP, PKIX-CRL (include in sslDecode ?)
- BACnet

F FAQ

This section answers some frequently asked questions.

F.1 If the hashtable is full, how much memory do I need to add?

When T2 warns you that the hashtable is full, it also tells you how to correct the problem:

```
[INF] Hash Autopilot: main HashMap full: flushing 1 oldest flow(s)! Fix: Invoke T2 with
                                    '-f 5' next time.
```

T2 calculates an estimate of the multiplication factor `HASHFACTOR` which you can set with the `-f` commandline option. By default the main hash autopilot is enabled which maintains the sanity of T2 even if it runs out of flow memory. Nevertheless, T2 will be faster if you feed him the recommended `-f` factor.

F.2 Can I change the timeout of a specific flow in my plugin?

That is possible because each flow owns a timeout value which can be altered even on packet basis. It enables the user to program stateful protocol plugins. Check out the tcpStates plugin as an inspiration.

F.3 Can I reduce the maximal flow length?

In *tranalyzer2/src/tranalyzer.h* you will find a constant called `FDURLIMIT`. Set it to the amount of seconds you like and T2 will terminate every flow with max `FDURLIMIT+1` seconds. And create a new flow for the next packet to come.

F.4 How can I change the separation character in the flow file?

The separation character is defined as `SEP_CHAR` in *utils/bin2txt.h*. It can be set to any character(s), e.g., `","` or `"||"`. In addition, the character(s) used for comments, e.g., column names, is controlled by `HDR_CHR` in the same file. Note that Tranalyzer default values are `"\t"` and `"%"`, respectively. Be advised that if you changed either of those values, some scripts may not work as expected.

F.5 How can I build all the plugins?

If you invoked the script setup.sh then you may use
`t2build -a`
otherwise, old school:

```
   cd  /tranalyzer2-0.8.4
./autogen.sh -a
```

F.6 T2 failed to compile: What can I do?

If a dependency is missing, you should see an appropriate messsage, e.g., *Missing dependency libname*. If no such message is displayed, it could be that the Makefiles are outdated. Then use `autogen.sh -r` to force the rebuild of the Makefiles. A typical error requiring the use of `autogen.sh -r` is:

```
...
/bin/bash: line 10: automake-: command not found
Makefile:333: recipe for target 'Makefile.in' failed
make[1]: *** [Makefile.in] Error 127
...
```

If you see the following message, then the autotools are not installed.

```
make: Entering directory '/home/user/tranalyzer2-0.8.4/tranalyzer2/doc'
make: Nothing to be done for 'clean'.
make: Leaving directory '/home/user/tranalyzer2-0.8.4/tranalyzer2/doc'
../autogen.sh: line 116: autoreconf: command not found
../autogen.sh: line 118: ./configure: No such file or directory

Failed to configure tranalyzer2
```

In this case, please refer to the *doc/tutorials/install.pdf*.

F.7 T2 segfaults: What can I do?

T2 never segfaults! Unless he deviates from his cosmic plan and indeed segfaults. The prominent reason are memory inconsistencies with old plugins being resident under `~/.tranalyzer/plugins/`.

1. Remove all the plugins: `rm ~/.tranalyzer/plugins/*.so`
2. Recompile the plugins, e.g., `cd ~/tranalyzer2-0.8.4/ && ./autogen.sh`
3. T2 should behave again.

For the developer:
If that does not fix the problem, recompile T2 in debug mode with `./autogen.sh -d` and try to run tranalyzer in *gdb*: `gdb -args ./tranalyzer -r file.pcap -w outpref`. If the error happens while writing flows, try to remove plugins until the error disappears. Finally, run the `segvtrack` script as follows: `segvtrack yourpcap`. This will automatically reduce the PCAP to the smallest set of packets which causes a segfault. If this does not help, send us a bug report at tranalyzer@rdit.ch with this pcap, T2 configuration (the values that differ from the default) and the plugins you are using. Then we will get a fix for you in no time.

F.8 socketSink plugin aborts with "could not connect to socket: Connection refused"

The socketSink plugins acts as a client in a socket communication. Therefore, a server listening to `SERVADD`, `DPORT` and `SOCKTYPE` is required. As described in the **Example** Section of the socketSink plugin documentation, a simple server can be set up with netcat as follows: `nc -l 127.0.0.1 6666`. Make sure the address and port match the values listed in *socketSink.h*.

F.9 T2 stalls after USR1 interrupt: What can I do?

It is a bug in the libpcap, which somehow is not thread-safe under certain conditions. Check whether T2 is set to default signal threading mode in (`main.h`):

- Set `MONINTTHRD` to 1
- Set `MONINTPSYNC` to 1

Do not forget to recompile T2 with `./autogen.sh` if you had to change the configuration.

Now the process of printing is detached from the packet capture and the output is synchronized to the packet processing main loop. Thus, pcap is never interrupted.

F.10 Can I reuse my configuration between different machines or Tranalyzer versions?

You can write a patch for t2conf and use it as follows: `t2conf --patch file.patch`. Revert the patch with the `--rpatch` option. The patch is a simple text file listing the defines to change, e.g., `IPV6_ACTIVATE <tab> 1 <tab> 0 <tab> tranalyzer2/src/networkHeaders.h`. For more details, refer to the documentation of t2conf .

F.11 How to contribute code, submit a bug or request a feature?

Contact the Anteater via email at tranalyzer@rdit.ch, and he will answer you.