
Tranalyzer2

socketSink



Output Into a TCP/UDP Socket



Tranalyzer Development Team

Contents

1	socketSink	1
1.1	Description	1
1.2	Dependencies	1
1.3	Configuration Flags	1
1.4	Additional Output	2
1.5	Example	3
1.6	Post-Processing	3
1.7	t2b2t	3

1 socketSink

1.1 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 visualizing 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. The socketSink plugin produces output directly into the Ethernet interface.

1.2 Dependencies

1.2.1 External Libraries

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

SKS_GZ_COMPRESS=1		
Ubuntu:	<code>sudo apt-get install</code>	<code>zlib1g-dev</code>
Arch:	<code>sudo pacman -S</code>	<code>zlib</code>
Gentoo:	<code>sudo emerge</code>	<code>zlib</code>
openSUSE:	<code>sudo zypper install</code>	<code>zlib-devel</code>
Red Hat/Fedora¹:	<code>sudo dnf install</code>	<code>zlib-devel</code>
macOS²:	<code>brew install</code>	<code>zlib</code>

1.2.2 Core Configuration

This plugin requires the following core configuration:

- `$T2HOME/tranalyzer2/src/tranalyzer.h`:

– `BLOCK_BUF=0`

1.3 Configuration Flags

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

Name	Default	Description	Flags
<code>SKS_SERVADD</code>	<code>"127.0.0.1"</code>	destination address	
<code>SKS_DPORT</code>	<code>6666</code>	destination port (host order)	
<code>SKS_SOCKETYPE</code>	<code>1</code>	Socket type: 0: UDP; 1: TCP	
<code>SKS_GZ_COMPRESS</code>	<code>0</code>	Compress (gzip) the output	<code>SKS_SOCKETYPE=1</code>
<code>SKS_CONTENT_TYPE</code>	<code>1</code>	0: binary; 1: text; 2: JSON	
<code>SKS_HOST_INFO</code>	<code>0</code>	0: no info; 1: all info about host	<code>SKS_CONTENT_TYPE=0</code>

¹If the `dnf` command could not be found, try with `yum` instead

²Brew is a packet manager for macOS that can be found here: <https://brew.sh>

1.3.1 bin2txt.h

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

Variable	Default	Description	Flags
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: 0: full 128 bit length 1: compressed	
TFS_EXTENDED_HEADER	0	Print an extended header in the flow file (number of rows, columns, columns type)	
B2T_LOCALTIME	0	Time representation: 0: UTC, 1: localtime	
B2T_NANOSECS	0	Time precision: 0: microsecs, 1: nanosecs	
HDR_CHR	"%"	start character of comments in flow file	
SEP_CHR	"\t"	character to use to separate the columns in the flow file	

1.3.2 Environment Variable Configuration Flags

The following configuration flags can also be configured with environment variables (`ENVCTRL>0`):

- `SKS_SERVADD`
- `SKS_DPORT`

1.4 Additional Output

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

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

Parameter	Type	Description	Flags
1	U32	Message length	BUF_DATA_SHFT>0
2	U32	Checksum	BUF_DATA_SHFT>1
3	U32	Sensor ID	
4	U64.U32	Present Unix timestamp	
5	RS	OS;Machine Name;built;OS type;HW;	
6	RS	Ethername(address) or IPInterfacename(address/netmask)	

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	Flags
1	U32	Message length	BUF_DATA_SHFT>0
2	U32	Checksum	BUF_DATA_SHFT>1
3	RU32	Binary buffer output	

1.5 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`

1.6 Post-Processing

1.7 t2b2t

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

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

The use of the program is straightforward:

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

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