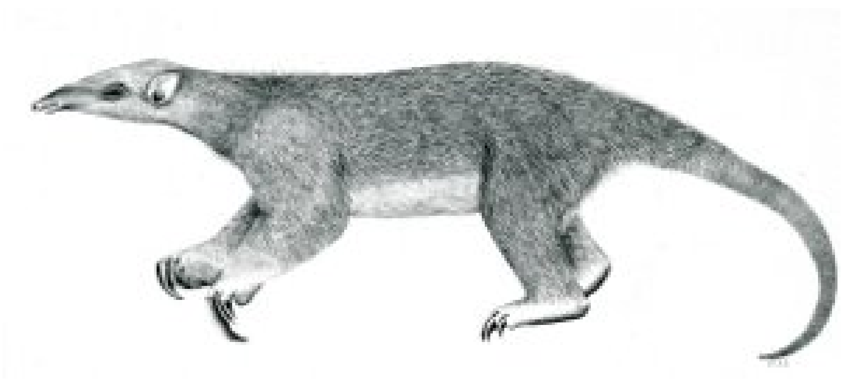
Tranalyzer2

socketSink

Output Into a TCP/UDP Socket



Tranalyzer Development Team

Contents

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 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.

1.2 Dependencies

1.2.1 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`[1]

1.3 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`

1.3.1 bin2txt.h

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

[1]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

1.4 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

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`