Table of Contents

Configuration and installation

The DaCoPAn Analyzer is commonly distributed in a source package. To install the software:

1. Unpack the archive.
2. Run the `configure` script to check your system configuration.
3. If the configure script complains about missing libraries or header files, install the required packages to satisfy the dependencies and run `configure` again. To compile the program you will need make, gcc ≥ 3.2 and libpcap ≥ 0.7.1 (earlier versions may work as well, but was not tested).
4. Run `make` command to build the program.
5. Run `make install` command to install the software.

The DaCoPAn Analyzer was only tested under Linux 2.4 and (most probably) will not compile for other operating systems.

For more details see the INSTALL file included with the source distribution.

Synopsis

dacopan-analyzer [*OPTIONS*] *FILE1 FILE2*

Description

The *dacopan-analyzer* is a part of *DaCoPAn* software.

The *dacopan-analyzer* program is used to read, merge and process data from tcpdump log files and store it in the protocol events file readable by the dacopan-animator.

All irrelevant information in log files are ignored. Each two corresponding packets are linked between log files (packet sent - packet received). For each non-linked packet *dacopan-analyzer* makes event "UNIT_DROPPED". All relevant information from log files are stored in the produced protocol events file.

Usually *dacopan-analyzer* is used only with -i flags. These flags are necessary for program and specifies used IP addresses.

The basic input data for *dacopan-analyzer* is two tcpdump log files which are created by *tcpdump* tool. The simple command for creating tcpdump logs is the following:

- Host 192.168.0.1 - `tcpdump -s 0 -w 1.dump host 192.168.0.2`
- Host 192.168.0.2 - `tcpdump -s 0 -w 2.dump host 192.168.0.1`

This examples also show tcpdump filter: the packets from other hosts does not include into produced log files.

The *dacopan-analyzer* has the following sequence of operations:

1. First user sets necessary options (IP addresses, file names) and run program.
2. *dacopan-analyzer* reads tcpdump log files and filters unnecessary data. Also using defined application port numbers defines application level.
3. Each message contains in two log files: in one hand - message sent and in another hand - message received. This parts are calls *events*. *dacopan-analyzer* build links between this events. Also *dacopan-analyzer* merges all events into one sequence.
4. The next step is split sequence between *flows*. Each flow is a quantity of messages between fixed ports. In this step *dacopan-analyzer* build events for all levels, defragments messages, makes "UNIT_DROPPED" events and calculates variables.
5. The last step is a write PEF file.

Basic version of *dacopan-analyzer* supports the following protocols:

- Arp level - ARP;
- Network level - IPv4;
- Transport level - TCP, UDP;
- Application level - HTTP, DNS.

The protocols for application level defines using port numbers. For each known application protocol *dacopan-analyzer* has one predefined port number. The events for other protocols includes as "UNKNOWN". User can manually adds other protocols to the produced PEF file.

Options

-i, **--ip-address**=*IP1[,IP2]*

> Specify IP addresses corresponding to packet trace files. This option requires the argument --- a list of two comma separated IP addresses in standard numbers-and-dots notation. There should be no white spaces inside the list. Alternatively, the option could be specified twice with single IP address as an argument.

-l, **--var-log**=*FILE*

> Read variables from FILE, containing some host-specific variables, which can not be determined from packet trace logs (not supported).

-o, **--output**=*FILE*

> Place the output into FILE. This option requires the argument --- the name of the output file. The default output file name is used if the option is not specified.

-t, **--time-align**=*VAL1[,VAL2]*

> Set time alignments (time difference) between the hosts.

--http=*PORT1[,...]*

Specify list of HTTP ports

--dns=*PORT1[,...]*

Specify list of DNS ports

-h, **--help**

Show brief documentation and exit

-v, **--version**

Show version information and exit

Protocol events file

The Protocol Events File (PEF) is the interface between Analyzer and Animator modules. It is the output of the Analyzer, it is read by the Animator, and it contains the necessary packet interchange data. Some of this information may as well have been added manually to the PEF. The use of XML has been preferred in an attempt to make the PEF extensible, human-readable and editable.

The *DTD* (Document Type Definition) to which any PEF must conform follows. It contains information about:

host

Hosts include an id for internal use, its IP address, and maybe a hostname.

flow

Flows include an id, 2 host ids, and their 2 ports.

link

Links include an id and 2 host ids.

layer

Layers include an id and name; inside each layer definition, the protocols particular to that layer can be found.

protocol

Protocol definition includes an id and a name.

variable

Variables contain constant and dynamic variable definitions.

constant

Constants include a name, and maybe a host id, a link id and/or a protocol id. The value of the constant variable is specified after the definition.

variable

Variables (those variables that change from unit to unit, for instance) include name, protocol id and scope (flow/unit/unit-field). Their values are specified later inside the units.

event

Events different types of units compose the actual packet data.

unit_sent

Unit sents include id, source id, destination id, protocol id and may include flow id, time and children id list. Values of dynamic variables are specified inside them.

unit_received

Unit receiveds include an id corresponding to a unit sent id, and the time when the unit was received in destination.

unit_dropped

Unit droppeds have an id and the time when the unit is dropped.

For more detailed information see the "events.dtd" file.

Examples

Analyzer usage

There are some examples of dacopan-analyzer usage:

- `dacopan-analyzer -v` --- print version of the Analyzer
- `dacopan-analyzer -i 10.0.0.1,10.0.0.2 dump1 dump2` --- create PEF file from the dump1 and dump2 files.
- `dacopan-analyzer -i 192.168.0.1 dump1 -i 192.168.0.2 dump2 --dns=36255 -t 0.130000` --- create PEF file, specify DNS port and time alignment.
- `dacopan-analyzer -i 192.168.0.1 dump1 -i 192.168.0.2 dump2 --http=3625` --- create PEF file, specify HTTP ports.
- `dacopan-analyzer -i 192.168.0.1 dump1 -i 192.168.0.2 dump2 -o`

`out.pef` --- create PEF file with out.pef name.

tcpdump log files and PEF file

The two tcpdump log files contains information about connection using UDP protocol. The textual presentation of these files is following.

dump1

```
16:37:33.673745 10.0.0.1 > 10.0.0.2: (frag 40657:532@976) (ttl 64, len 552)
16:37:33.874760 10.0.0.1.36568 > 10.0.0.2.36264: [bad udp cksum 2f1!] udp 1500 (frag 40657:976@0+) (ttl 64, len 996)
16:37:35.676990 10.0.0.1 > 10.0.0.2: (frag 40658:556@1952) (ttl 64, len 576)
16:37:35.878496 10.0.0.1 > 10.0.0.2: (frag 40658:976@976+) (ttl 64, len 996)
16:37:35.879481 10.0.0.1.36569 > 10.0.0.2.36265: [bad udp cksum 404a!] udp 2500 (frag 40658:976@0+) (ttl 64, len 996)
```

dump2

```
16:37:33.567000 10.0.0.1 > 10.0.0.2: (frag 40657:532@976) (ttl 64, len 552)
16:37:33.567006 10.0.0.1.36568 > 10.0.0.2.36264: [bad udp cksum 2f1!] udp 1500 (frag 40657:976@0+) (ttl 64, len 996)
16:37:35.569742 10.0.0.1 > 10.0.0.2: (frag 40658:556@1952) (ttl 64, len 576)
16:37:35.569748 10.0.0.1 > 10.0.0.2: (frag 40658:976@976+) (ttl 64, len 996)
16:37:35.569892 10.0.0.1.36569 > 10.0.0.2.36265: [bad udp cksum 404a!] udp 2500 (frag 40658:976@0+) (ttl 64, len 996)
```

Using dacopan-analyzer we produce the following PEF file.

PEF file

```
<?xml version="1.0"?>
<!DOCTYPE protocol_events SYSTEM "http://www.cs.helsinki.fi/group/dacopan/events.dtd">

<!--
    Created by: dacopan-analyzer version 1.0rc1.
    Tue May 25 18:33:03 2004
-->

<protocol_events>

    <hosts>
        <host id="H1" ip="10.0.0.2"/>
        <host id="H2" ip="10.0.0.1"/>
    </hosts>

    <flows>
        <flow id="F1" host1="H1" port1="36568" host2="H2" port2="36264"/>
        <flow id="F2" host1="H1" port1="36569" host2="H2" port2="36265"/>
    </flows>

    <links>
        <link id="K1" host1="H1" host2="H2"/>
    </links>

    <layers>
        <layer id="L1" name="network">
            <protocol id="P2" name="net_unknown"/>
            <protocol id="P0" name="ipv4"/>
        </layer>
        <layer id="L2" name="transport">
            <protocol id="P5" name="trans_unknown"/>
            <protocol id="P3" name="tcp"/>
            <protocol id="P4" name="udp"/>
        </layer>
        <layer id="L3" name="application">
            <protocol id="P8" name="application_unknown"/>
            <protocol id="P7" name="http"/>
            <protocol id="P6" name="dns"/>
        </layer>
```

```
</layers>

<variables>
    <variable name="tos" protocol="P0" scope="unit-field"/>
    <variable name="tot_len" protocol="P0" scope="unit-field"/>
    <variable name="id" protocol="P0" scope="unit-field"/>
    <variable name="flag_rf" protocol="P0" scope="unit-field"/>
    <variable name="flag_df" protocol="P0" scope="unit-field"/>
    <variable name="flag_mf" protocol="P0" scope="unit-field"/>
    <variable name="frag_off" protocol="P0" scope="unit-field"/>
    <variable name="ttl" protocol="P0" scope="unit-field"/>
    <variable name="protocol" protocol="P0" scope="unit-field"/>
    <variable name="source_addr" protocol="P0" scope="unit-field"/>
    <variable name="dest_addr" protocol="P0" scope="unit-field"/>
    <variable name="source_port" protocol="P3" scope="unit-field"/>
    <variable name="dest_port" protocol="P3" scope="unit-field"/>
    <variable name="seq" protocol="P3" scope="unit-field"/>
    <variable name="ack_seq" protocol="P3" scope="unit-field"/>
    <variable name="window" protocol="P3" scope="unit-field"/>
    <variable name="urg_pointer" protocol="P3" scope="unit-field"/>
    <variable name="flag_fin" protocol="P3" scope="unit-field"/>
    <variable name="flag_syn" protocol="P3" scope="unit-field"/>
    <variable name="flag_rst" protocol="P3" scope="unit-field"/>
    <variable name="flag_psh" protocol="P3" scope="unit-field"/>
    <variable name="flag_ack" protocol="P3" scope="unit-field"/>
    <variable name="flag_urg" protocol="P3" scope="unit-field"/>
    <variable name="data_offset" protocol="P3" scope="unit-field"/>
    <variable name="source_port" protocol="P4" scope="unit-field"/>
    <variable name="dest_port" protocol="P4" scope="unit-field"/>
    <variable name="len" protocol="P4" scope="unit-field"/>
    <variable name="method" protocol="P7" scope="unit-field"/>
    <variable name="url" protocol="P7" scope="unit-field"/>
    <variable name="version" protocol="P7" scope="unit-field"/>
    <variable name="status_code" protocol="P7" scope="unit-field"/>
    <variable name="phrase" protocol="P7" scope="unit-field"/>
    <variable name="connection" protocol="P7" scope="unit-field"/>
    <variable name="content_length" protocol="P7" scope="unit-field"/>
    <variable name="content_type" protocol="P7" scope="unit-field"/>
    <variable name="cookie" protocol="P7" scope="unit-field"/>
    <variable name="date" protocol="P7" scope="unit-field"/>
    <variable name="host" protocol="P7" scope="unit-field"/>
    <variable name="user_agent" protocol="P7" scope="unit-field"/>
    <variable name="trans_time" protocol="P2 P0 P5 P3 P4 P8 P7" scope="unit"/>
    <variable name="sent_time" protocol="P2 P0 P5 P3 P4 P8 P7" scope="unit"/>
    <variable name="data" protocol="P8 P7" scope="unit"/>
</variables>

<events>
    <unit_sent id="U1" source="H2" destination="H1"
     protocol="P8" time="0.000000" children="U2" flow="F1">
        <value name="sent_time">0.000000</value>
        <value name="trans_time">0.307760</value>
        <value name="data"><![CDATA[...]]></value>
    </unit_sent>
    <unit_sent id="U2" source="H2" destination="H1"
     protocol="P4" time="0.000000" children="U3 U4" flow="F1">
        <value name="sent_time">0.000000</value>
        <value name="trans_time">0.307760</value>
        <value name="source_port">36568</value>
        <value name="dest_port">36264</value>
        <value name="len">1508</value>
    </unit_sent>
    <unit_sent id="U4" source="H2" destination="H1"
```

```
 protocol="P0" time="0.000000">
    <value name="sent_time">0.000000</value>
    <value name="trans_time">0.106745</value>
    <value name="tos">0</value>
    <value name="tot_len">552</value>
    <value name="id">40657</value>
    <value name="flag_rf">0</value>
    <value name="flag_df">1</value>
    <value name="flag_mf">0</value>
    <value name="frag_off">976</value>
    <value name="ttl">64</value>
    <value name="protocol">17</value>
    <value name="check">34167</value>
    <value name="source_addr">10.0.0.1</value>
    <value name="dest_addr">10.0.0.2</value>
</unit_sent>
<unit_sent id="U3" source="H2" destination="H1"
 protocol="P0" time="0.000006">
    <value name="sent_time">0.000006</value>
    <value name="trans_time">0.307754</value>
    <value name="tos">0</value>
    <value name="tot_len">996</value>
    <value name="id">40657</value>
    <value name="flag_rf">0</value>
    <value name="flag_df">1</value>
    <value name="flag_mf">1</value>
    <value name="frag_off">0</value>
    <value name="ttl">64</value>
    <value name="protocol">17</value>
    <value name="check">25653</value>
    <value name="source_addr">10.0.0.1</value>
    <value name="dest_addr">10.0.0.2</value>
</unit_sent>
<unit_received id="U4" time="0.106745"/>
<unit_received id="U3" time="0.307760"/>
<unit_received id="U2" time="0.307760"/>
<unit_received id="U1" time="0.307760"/>
<unit_sent id="U5" source="H2" destination="H1"
 protocol="P8" time="2.002742" children="U6" flow="F2">
    <value name="sent_time">2.002742</value>
    <value name="trans_time">0.309739</value>
    <value name="data"><![CDATA[...]]></value>
</unit_sent>
<unit_sent id="U6" source="H2" destination="H1"
 protocol="P4" time="2.002742" children="U7 U8 U9" flow="F2">
    <value name="sent_time">2.002742</value>
    <value name="trans_time">0.309739</value>
    <value name="source_port">36569</value>
    <value name="dest_port">36265</value>
    <value name="len">2508</value>
</unit_sent>
<unit_sent id="U9" source="H2" destination="H1"
 protocol="P0" time="2.002742">
    <value name="sent_time">2.002742</value>
    <value name="trans_time">0.107248</value>
    <value name="tos">0</value>
    <value name="tot_len">576</value>
    <value name="id">40658</value>
    <value name="flag_rf">0</value>
    <value name="flag_df">1</value>
    <value name="flag_mf">0</value>
    <value name="frag_off">1952</value>
    <value name="ttl">64</value>
```

```
            <value name="protocol">17</value>
            <value name="check">34020</value>
            <value name="source_addr">10.0.0.1</value>
            <value name="dest_addr">10.0.0.2</value>
        </unit_sent>
        <unit_sent id="U8" source="H2" destination="H1"
         protocol="P0" time="2.002748">
            <value name="sent_time">2.002748</value>
            <value name="trans_time">0.308748</value>
            <value name="tos">0</value>
            <value name="tot_len">996</value>
            <value name="id">40658</value>
            <value name="flag_rf">0</value>
            <value name="flag_df">1</value>
            <value name="flag_mf">1</value>
            <value name="frag_off">976</value>
            <value name="ttl">64</value>
            <value name="protocol">17</value>
            <value name="check">25530</value>
            <value name="source_addr">10.0.0.1</value>
            <value name="dest_addr">10.0.0.2</value>
        </unit_sent>
        <unit_sent id="U7" source="H2" destination="H1"
         protocol="P0" time="2.002892">
            <value name="sent_time">2.002892</value>
            <value name="trans_time">0.309589</value>
            <value name="tos">0</value>
            <value name="tot_len">996</value>
            <value name="id">40658</value>
            <value name="flag_rf">0</value>
            <value name="flag_df">1</value>
            <value name="flag_mf">1</value>
            <value name="frag_off">0</value>
            <value name="ttl">64</value>
            <value name="protocol">17</value>
            <value name="check">25652</value>
            <value name="source_addr">10.0.0.1</value>
            <value name="dest_addr">10.0.0.2</value>
        </unit_sent>
        <unit_received id="U9" time="2.109990"/>
        <unit_received id="U8" time="2.311496"/>
        <unit_received id="U7" time="2.312481"/>
        <unit_received id="U6" time="2.312481"/>
        <unit_received id="U5" time="2.312481"/>
    </events>

</protocol_events>
```

Frequently Asked Questions

How can I produce PEF file?
Why produced PEF files does not have events?
The hosts has a different time. How I can correct it?

How can I produce PEF file?

You need two hosts and tcpdump tool in each host. You can use the following sequence:

1. first start tcpdump in each host with necessary options (for example filters unwanted data)
2. use the network between hosts (We belive that you know how)
3. stop tcpdump tools
4. using analyzer produce PEF file.

Why produced PEF files does not have events?

It seems that you use empty tcpdump log files or wrong IP addresses. Try to use tcpdump tool and you can see what packets are includes in this log files and what IP addresses are used.

The hosts has a different time. How I can correct it?

You can use option -t to increase or decrease time value for each host. Also if analyzer finds packet which was received before sent then it prints message with necessary time addition. But sometimes time grows irregularly and if tcpdump logs are large then is possible that analyzer can not links this logs. You can divide this logs between small parts and makes several short PEF files.