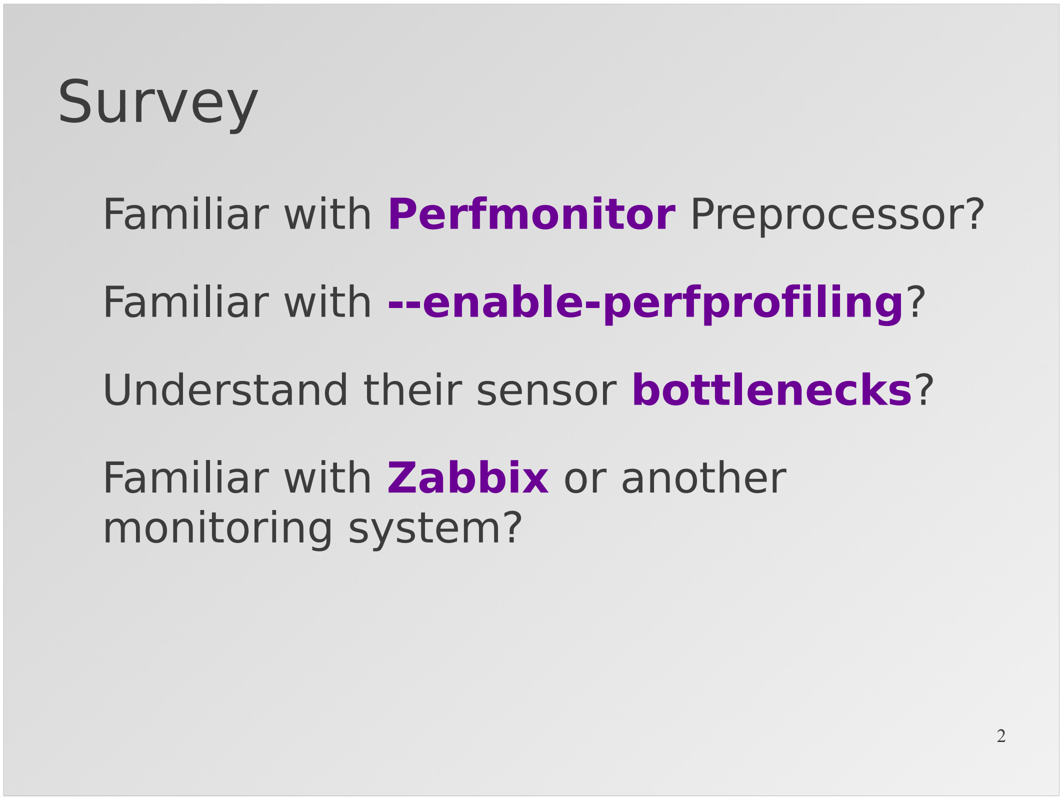
# Understanding Snort Performance

**An Evidence-Based Approach**

- Mike Lococo
- Senior Network Security Analyst
- New York University, largest private university in US
- <40k for-credit students +12k non-credit
- >15k employees
- Decentralized IT reporting structure
- 5 security staff



## Survey

Familiar with **Perfmonitor** Preprocessor?

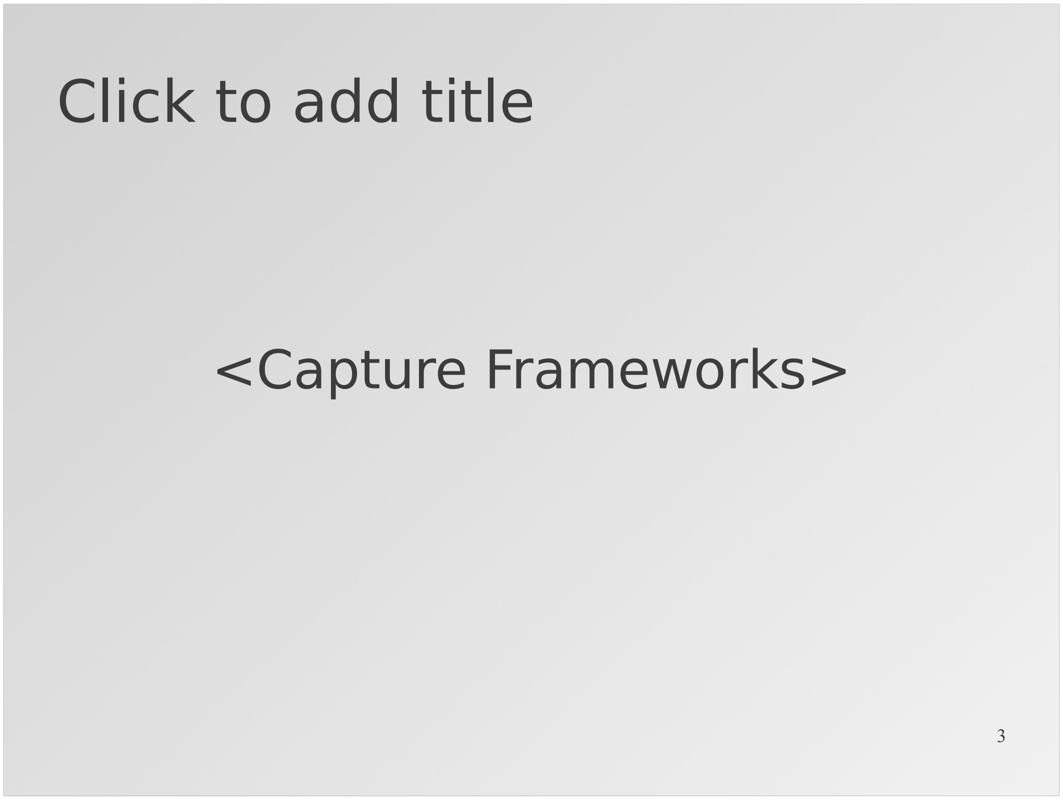
Familiar with **--enable-perfprofiling**?

Understand their sensor **bottlenecks**?

Familiar with **Zabbix** or another monitoring system?

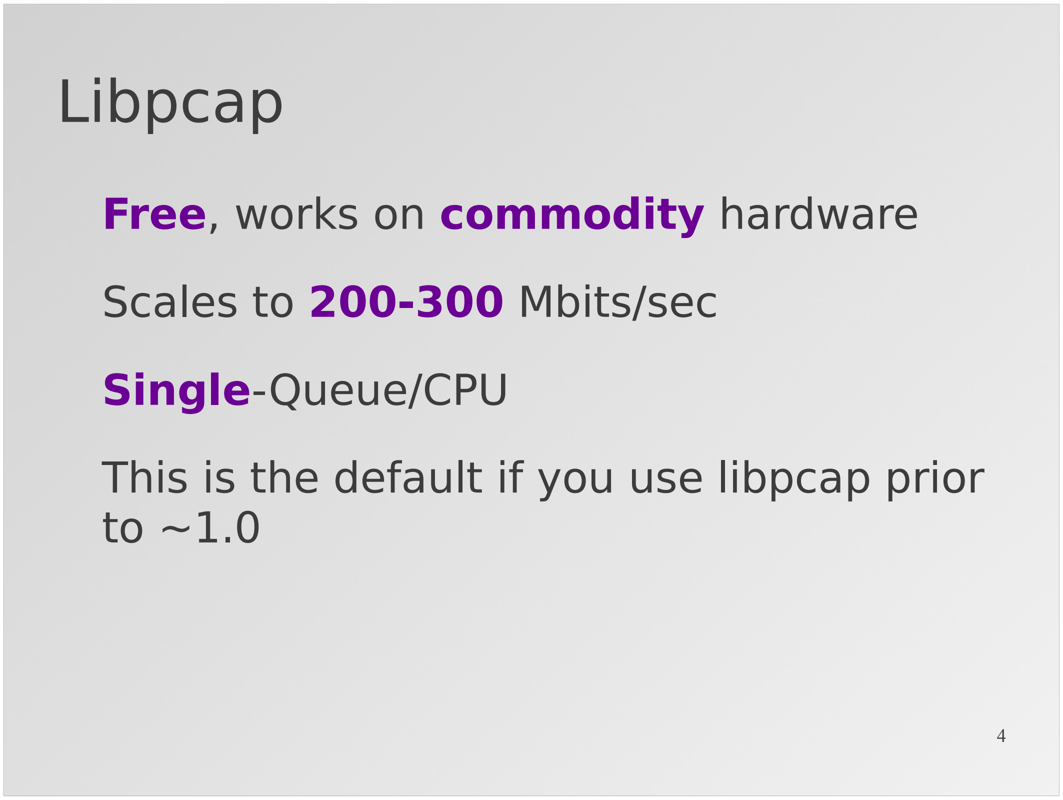
Snort + graphs is better than snort without graphs, that's pretty much all we're going to cover. You'll still make all the same dumb mistakes, they'll just be easier to find.

We'll cover snort instrumentation, combine it with some basic OS and other instrumentation, and then look at graphs. It's not rocket science, but it makes it easier to develop a situational awareness of what is “normal” for your environment.



# Click to add title

<Capture Frameworks>

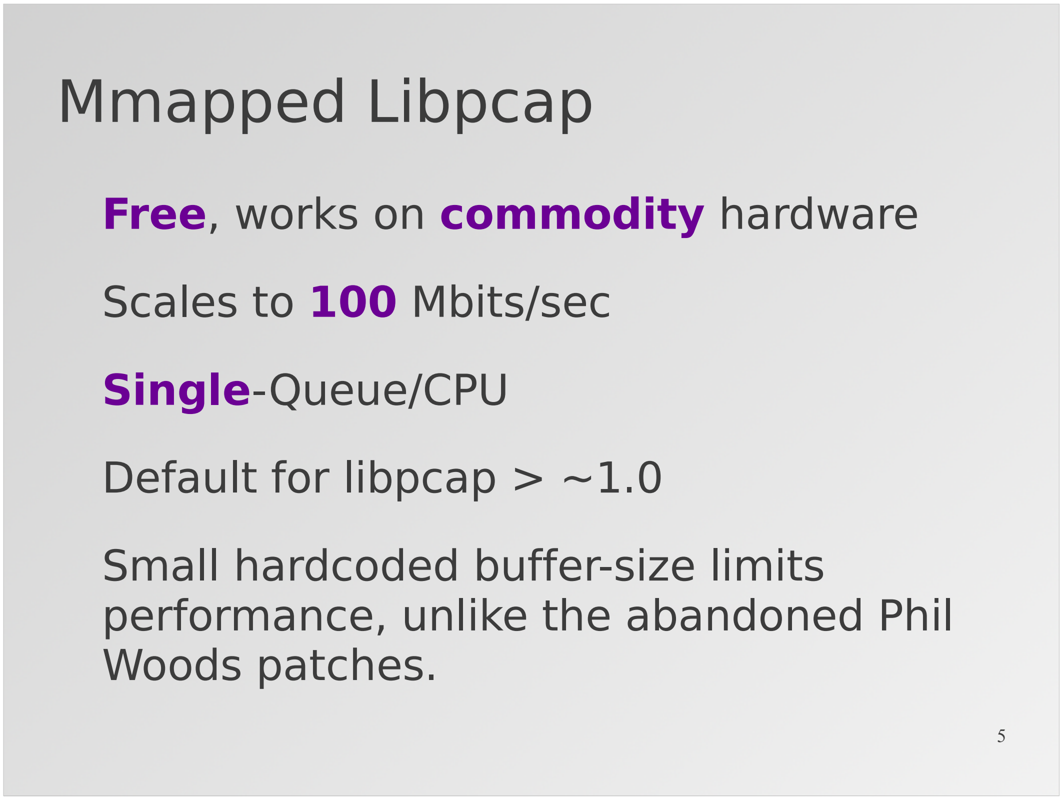


## Libpcap

- **Free**, works on **commodity** hardware
- Scales to **200-300** Mbits/sec
- **Single**-Queue/CPU
- This is the default if you use libpcap prior to ~1.0

Tweaks to sysctl.conf may help:

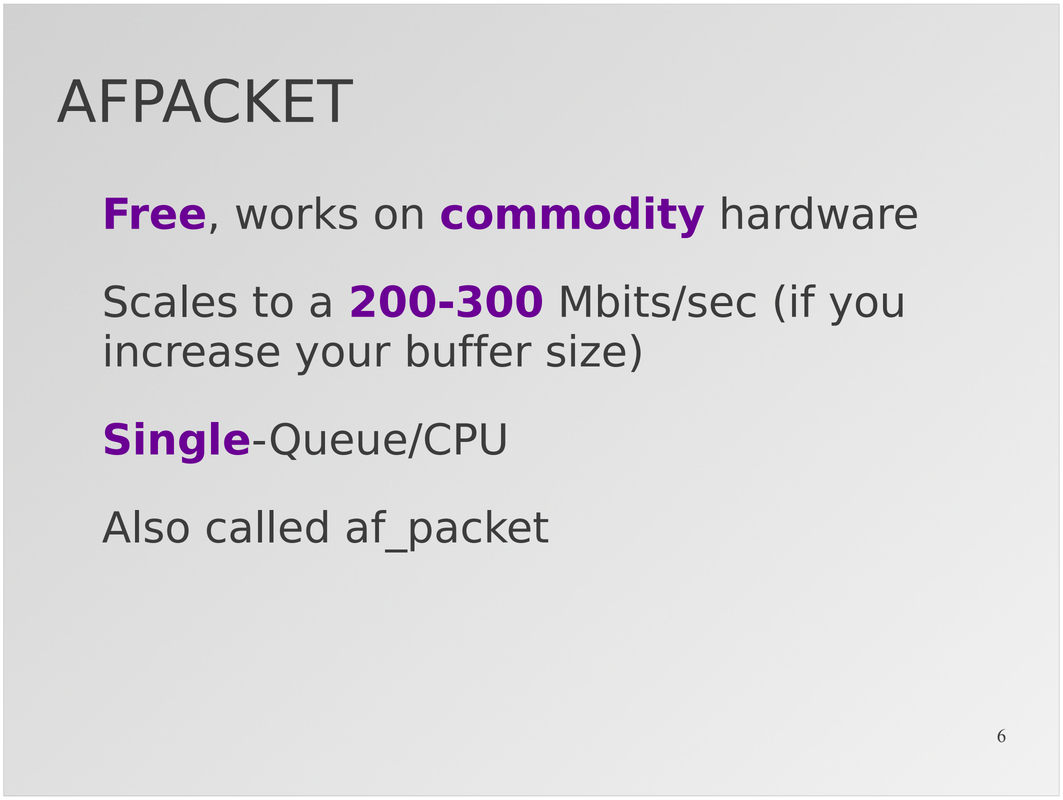
```
net.core.rmem_max = 33554432
net.core.netdev_max_backlog = 10000
net.core.rmem_default = 33554432
```



## Mmapped Libpcap

- **Free**, works on **commodity** hardware
- Scales to **100** Mbits/sec
- **Single**-Queue/CPU
- Default for libpcap > ~1.0
- Small hardcoded buffer-size limits performance, unlike the abandoned Phil Woods patches.

Note that performance may be **worse** with later libpcaps.



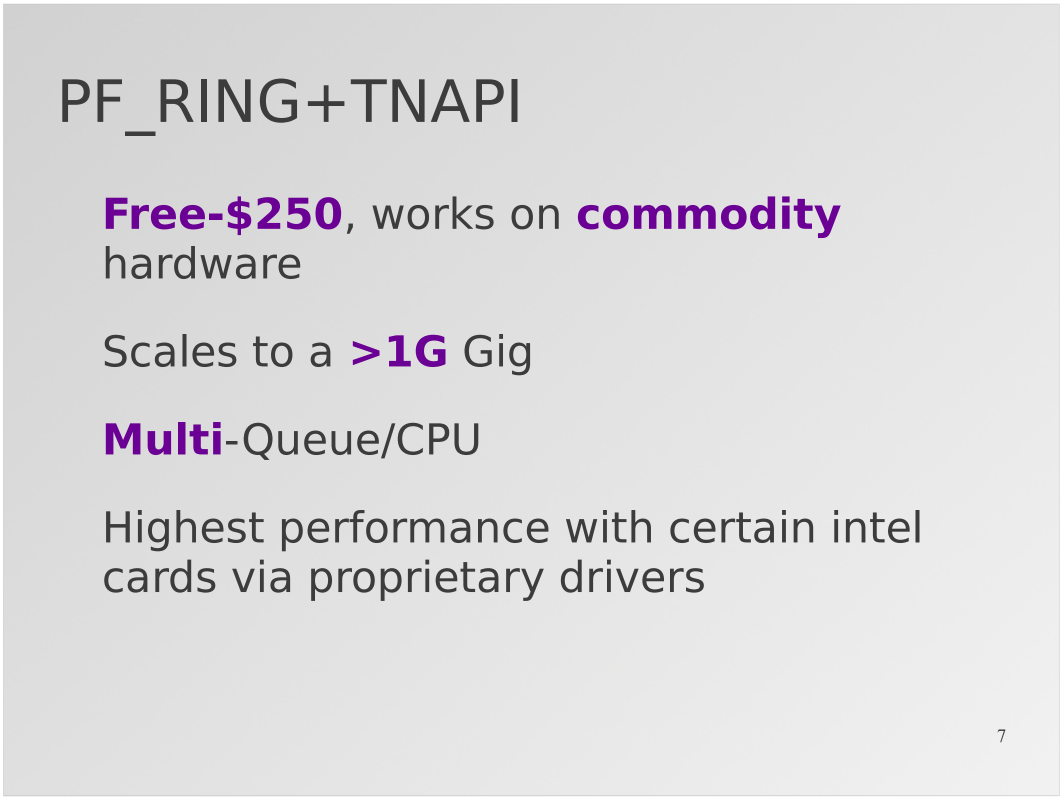
## AFPACKET

- **Free**, works on **commodity** hardware
- Scales to a **200-300** Mbits/sec (if you increase your buffer size)
- **Single**-Queue/CPU
- Also called af_packet

Snort Manual:
1.5.1 for syntax to configure from snort.conf
1.5.3 for syntax to configure from command-line

Sourcefire Howto:
http://vrt-blog.snort.org/2010/08/snort-29-essentials-daq.html

Kernel Interface Manpage for af_packet:
http://manpages.ubuntu.com/manpages/jaunty/man7/packet.7.html



## PF_RING+TNAPI

**Free-$250**, works on **commodity** hardware

Scales to a **>1G** Gig

**Multi**-Queue/CPU

Highest performance with certain intel cards via proprietary drivers

Explanation of TNAPI at ntop.org: http://www.ntop.org/TNAPI.html

Buy the high-performance drivers: http://www.ntop.org/shop/cart.php

Might scale close to 10gig, but no independent reports that I'm aware of confirm the ntop.org numbers.

# Intel X520 and Myricom 10G

**$1k-2k** dedicated **capture cards**

Scales to **10Gig**

**Multi**-Queue/CPU

Most reports are preliminary, these are relatively new cards. Linux drivers are supposed to be in the kernel.

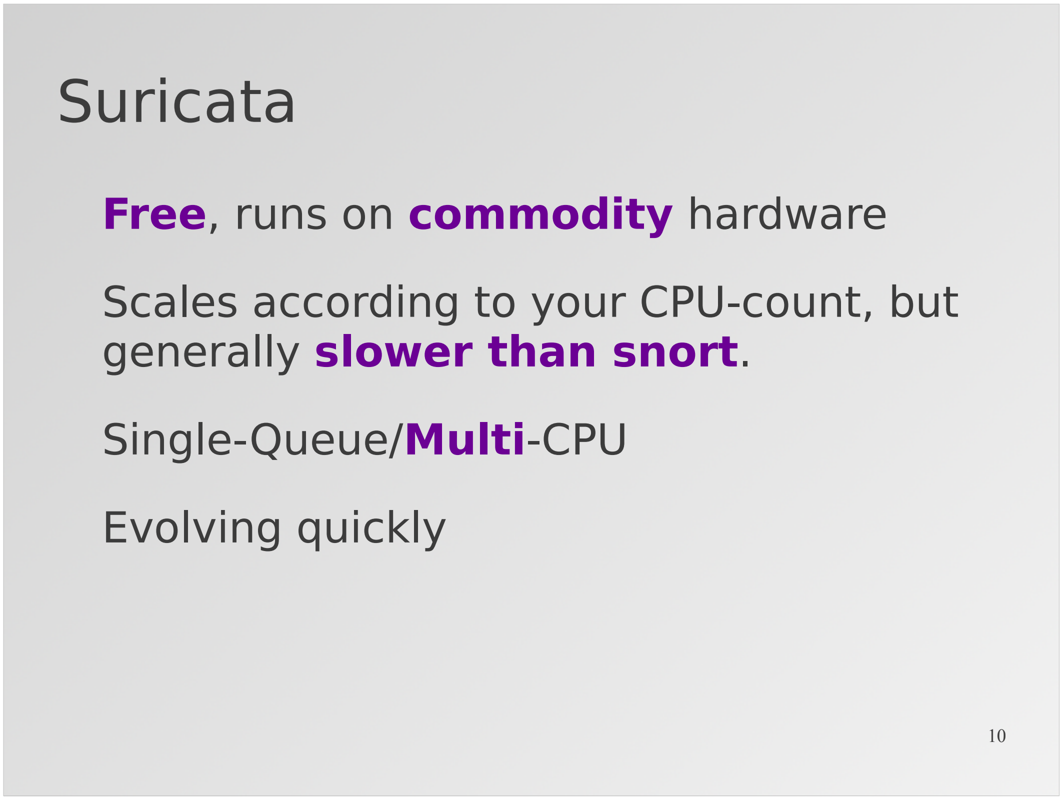
# Endace/Napatech

**$10k-20k** dedicated **capture cards**

Scales to **10Gig**

**Multi**-Queue/CPU

Well-established, well-tested. Expensive and a bit of a pain to manage.



## Suricata

**Free**, runs on **commodity** hardware

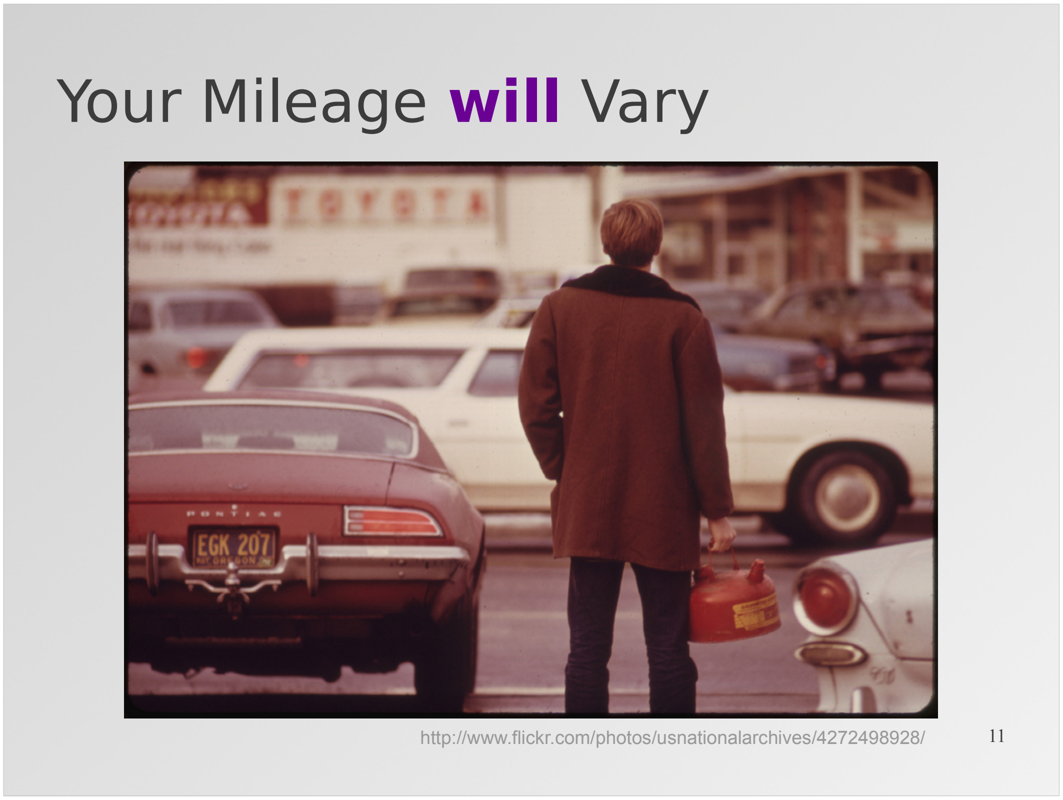
Scales according to your CPU-count, but generally **slower than snort**.

Single-Queue/**Multi**-CPU

Evolving quickly

Holisticinfosec Performance Test from August 2010: http://holisticinfosec.org/toolsmith/docs/august2010.html

Summary is that Suricata is 4x less CPU-efficient than snort, but if you compare a large multi-cpu Suricata instance to a single-cpu snort instance it can be faster given sufficient hardware.



## Your Mileage **will** Vary

http://www.flickr.com/photos/usnationalarchives/4272498928/

These are all anecdotes.

There are many site-specific factors, and you'll have to test locally to determine what works for you.

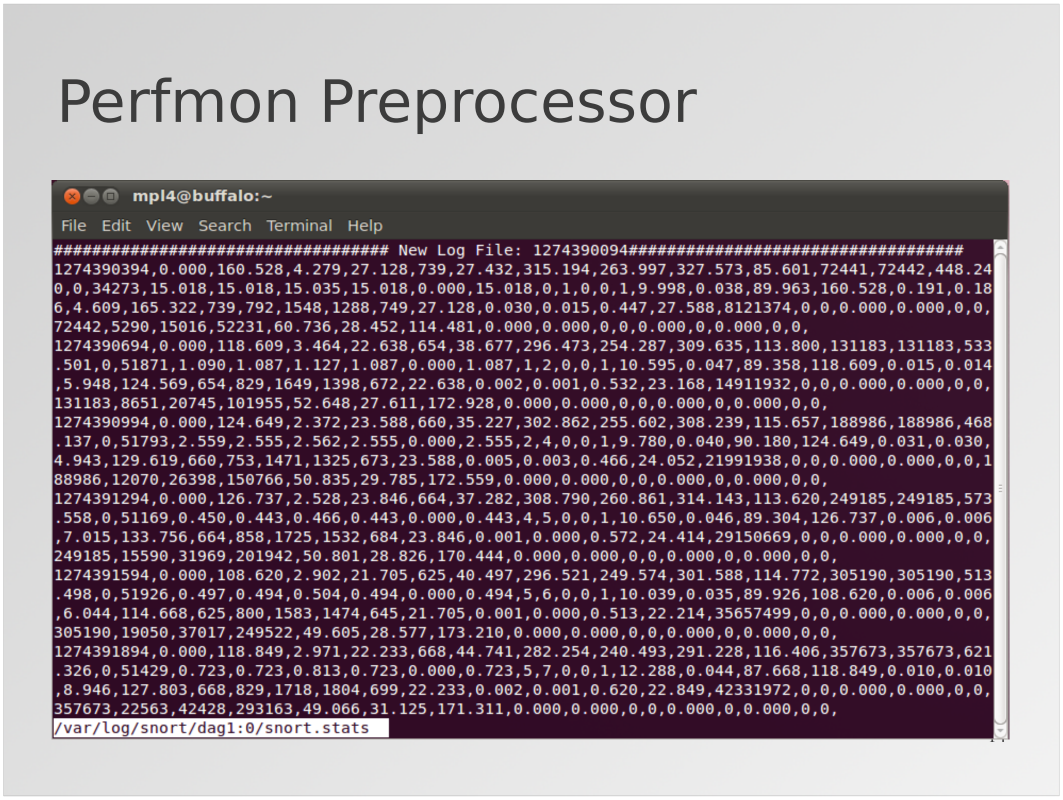
The remainder of this presentation is aimed at giving you the data you need to perform that local assessment.

# Click to add title

</Capture Frameworks>

# Click to add title

<Snort Instrumentation>



## Perfmon Preprocessor

```
################################## New Log File: 1274390094##################################
1274390394,0.000,160.528,4.279,27.128,739,27.432,315.194,263.997,327.573,85.601,72441,72442,448.24
0,0,34273,15.018,15.018,15.035,15.018,0.000,15.018,0,1,0,0,1,9.998,0.038,89.963,160.528,0.191,0.18
6,4.609,165.322,739,792,1548,1288,749,27.128,0.030,0.015,0.447,27.588,8121374,0,0,0.000,0.000,0,0,
72442,5290,15016,52231,60.736,28.452,114.481,0.000,0.000,0,0,0.000,0,0.000,0,0,
1274390694,0.000,118.609,3.464,22.638,654,38.677,296.473,254.287,309.635,113.800,131183,131183,533
.501,0,51871,1.090,1.087,1.127,1.087,0.000,1.087,1,2,0,0,1,10.595,0.047,89.358,118.609,0.015,0.014
,5.948,124.569,654,829,1649,1398,672,22.638,0.002,0.001,0.532,23.168,14911932,0,0,0.000,0.000,0,0,
131183,8651,20745,101955,52.648,27.611,172.928,0.000,0.000,0,0,0.000,0,0.000,0,0,
1274390994,0.000,124.649,2.372,23.588,660,35.227,302.862,255.602,308.239,115.657,188986,188986,468
.137,0,51793,2.559,2.555,2.562,2.555,0.000,2.555,2,4,0,0,1,9.780,0.040,90.180,124.649,0.031,0.030,
4.943,129.619,660,753,1471,1325,673,23.588,0.005,0.003,0.466,24.052,21991938,0,0,0.000,0.000,0,0,1
88986,12070,26398,150766,50.835,29.785,172.559,0.000,0.000,0,0,0.000,0,0.000,0,0,
1274391294,0.000,126.737,2.528,23.846,664,37.282,308.790,260.861,314.143,113.620,249185,249185,573
.558,0,51169,0.450,0.443,0.466,0.443,0.000,0.443,4,5,0,0,1,10.650,0.046,89.304,126.737,0.006,0.006
,7.015,133.756,664,858,1725,1532,684,23.846,0.001,0.000,0.572,24.414,29150669,0,0,0.000,0.000,0,0,
249185,15590,31969,201942,50.801,28.826,170.444,0.000,0.000,0,0,0.000,0,0.000,0,0,
1274391594,0.000,108.620,2.902,21.705,625,40.497,296.521,249.574,301.588,114.772,305190,305190,513
.498,0,51926,0.497,0.494,0.504,0.494,0.000,0.494,5,6,0,0,1,10.039,0.035,89.926,108.620,0.006,0.006
,6.044,114.668,625,800,1583,1474,645,21.705,0.001,0.000,0.513,22.214,35657499,0,0,0.000,0.000,0,0,
305190,19050,37017,249522,49.605,28.577,173.210,0.000,0.000,0,0,0.000,0,0.000,0,0,
1274391894,0.000,118.849,2.971,22.233,668,44.741,282.254,240.493,291.228,116.406,357673,357673,621
.326,0,51429,0.723,0.723,0.813,0.723,0.000,0.723,5,7,0,0,1,12.288,0.044,87.668,118.849,0.010,0.010
,8.946,127.803,668,829,1718,1804,699,22.233,0.002,0.001,0.620,22.849,42331972,0,0,0.000,0.000,0,0,
357673,22563,42428,293163,49.066,31.125,171.311,0.000,0.000,0,0,0.000,0,0.000,0,0,
/var/log/snort/dag1:0/snort.stats
```

CSV output.

Simple to configure.

Little/no cpu overhead to collect.

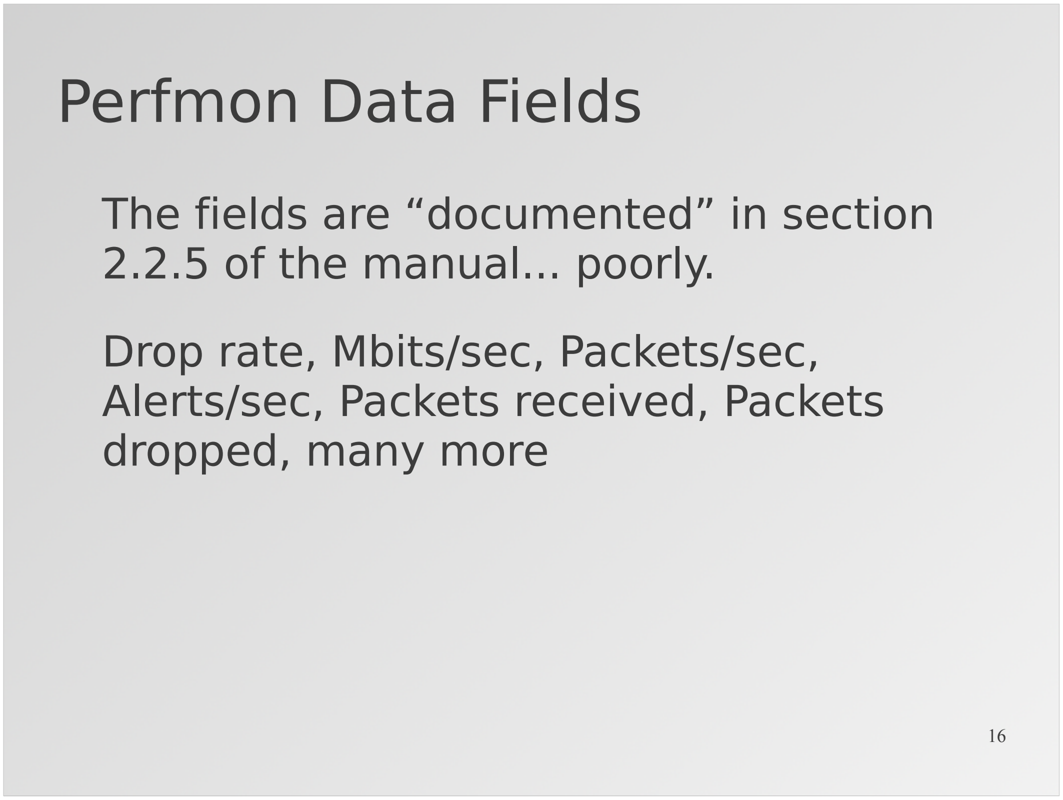
Rotate with logrotate, or just let it grow to a few hundred meg over the lifetime of your sensor.

Not much fun to interpret unless you have some way to process it beyond reading the CSV file.

# Configuring Perfmon

In Snort.conf:

```
preprocessor perfmonitor: time 300
file /var/log/snort/bogus.stats
pktcnt 10000
```



## Perfmon Data Fields

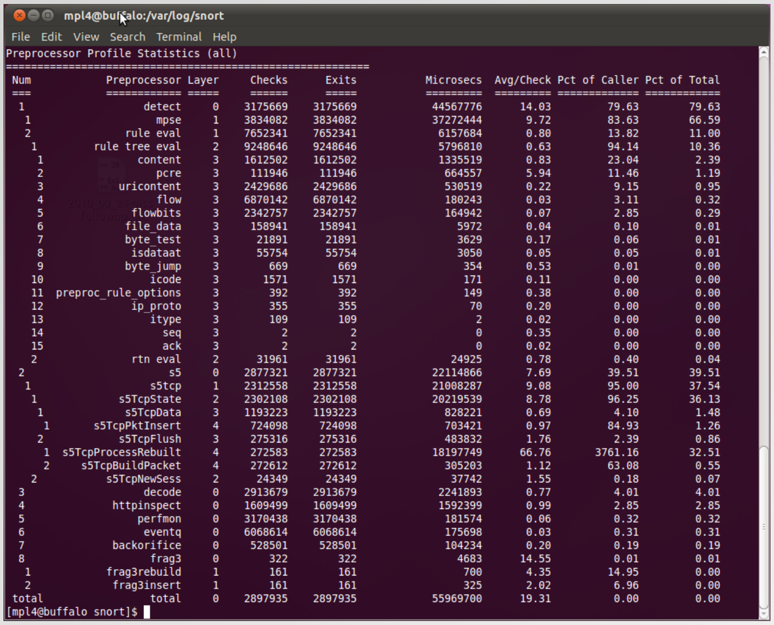
The fields are “documented” in section 2.2.5 of the manual... poorly.

Drop rate, Mbits/sec, Packets/sec, Alerts/sec, Packets received, Packets dropped, many more

Be aware that the drop-rate is averaged over the lifetime of the snort-process, not over the data-collection period. A 5% drop rate may be 5% over 24 hours, or 60% for 2 hours.

Other fields are generally averaged over the data-collection period, but one needs to be careful.

# Perfprofiling Preprocessors


mpl4@buffalo:/var/log/snort
File Edit View Search Terminal Help

```
Preprocessor Profile Statistics (all)
==========================================================
 Num            Preprocessor Layer     Checks      Exits           Microsecs  Avg/Check Pct of Caller Pct of Total
 ===            ============ =====     ======      =====           =========  ========= ============= ============
  1                   detect     0    3175669    3175669            44567776      14.03         79.63        79.63
   1                    mpse     1    3834082    3834082            37272444       9.72         83.63        66.59
   2               rule eval     1    7652341    7652341             6157684       0.80         13.82        11.00
    1         rule tree eval     2    9248646    9248646             5796810       0.63         94.14        10.36
     1               content     3    1612502    1612502             1335519       0.83         23.04         2.39
     2                  pcre     3     111946     111946              664557       5.94         11.46         1.19
     3            uricontent     3    2429686    2429686              530519       0.22          9.15         0.95
     4                  flow     3    6870142    6870142              180243       0.03          3.11         0.32
     5              flowbits     3    2342757    2342757              164942       0.07          2.85         0.29
     6             file_data     3     158941     158941                5972       0.04          0.10         0.01
     7             byte_test     3      21891      21891                3629       0.17          0.06         0.01
     8              isdataat     3      55754      55754                3050       0.05          0.05         0.01
     9             byte_jump     3        669        669                 354       0.53          0.01         0.00
    10                 icode     3       1571       1571                 171       0.11          0.00         0.00
    11  preproc_rule_options     3        392        392                 149       0.38          0.00         0.00
    12              ip_proto     3        355        355                  70       0.20          0.00         0.00
    13                 itype     3        109        109                   2       0.02          0.00         0.00
    14                   seq     3          2          2                   0       0.35          0.00         0.00
    15                   ack     3          2          2                   0       0.02          0.00         0.00
    2               rtn eval     2      31961      31961               24925       0.78          0.40         0.04
  2                       s5     0    2877321    2877321            22114866       7.69         39.51        39.51
   1                   s5tcp     1    2312558    2312558            21008287       9.08         95.00        37.54
    1             s5TcpState     2    2302108    2302108            20219539       8.78         96.25        36.13
     1             s5TcpData     3    1193223    1193223              828221       0.69          4.10         1.48
      1       s5TcpPktInsert     4     724098     724098              703421       0.97         84.93         1.26
     2            s5TcpFlush     3     275316     275316              483832       1.76          2.39         0.86
      1  s5TcpProcessRebuilt     4     272583     272583            18197749      66.76       3761.16        32.51
      2     s5TcpBuildPacket     4     272612     272612              305203       1.12         63.08         0.55
    2           s5TcpNewSess     2      24349      24349               37742       1.55          0.18         0.07
  3                   decode     0    2913679    2913679             2241893       0.77          4.01         4.01
  4              httpinspect     0    1609499    1609499             1592399       0.99          2.85         2.85
  5                  perfmon     0    3170438    3170438              181574       0.06          0.32         0.32
  6                   eventq     0    6068614    6068614              175698       0.03          0.31         0.31
  7              backorifice     0     528501     528501              104234       0.20          0.19         0.19
  8                    frag3     0        322        322                4683      14.55          0.01         0.01
   1            frag3rebuild     1        161        161                 700       4.35         14.95         0.00
   2             frag3insert     1        161        161                 325       2.02          6.96         0.00
 total                 total     0    2897935    2897935            55969700      19.31          0.00         0.00
[mpl4@buffalo snort]$ 
```

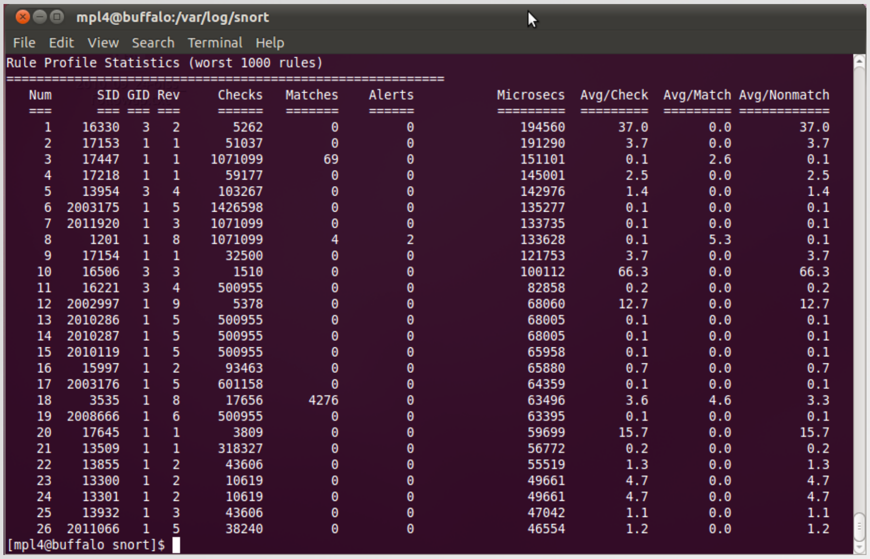
“Num” column shows component nesting within the snort process. For example, “mpse” and “rule eval” are components of the “detect” module.

“Percent of total” column gives a rough idea of how much cpu time that component takes up. It's not precise (note the top-level components do not add up to 100) but relative comparisons are useful.

In this sample, the pattern-matcher and the stream-preprocessor take up the vast-majority of time. This is relatively healthy for a snort instance with lots (~7k) rules enabled.

This snort can't be effectively tuned by stripping individual rules, we'd need to cut down the number to reduce the pattern matcher (mpse) CPU usage.

# Perfprofiling Rules



mpl4@buffalo:/var/log/snort
File Edit View Search Terminal Help

```
Rule Profile Statistics (worst 1000 rules)
==========================================================
   Num      SID GID Rev     Checks  Matches    Alerts           Microsecs  Avg/Check  Avg/Match Avg/Nonmatch
   ===      === === ===     ======  =======    ======           =========  =========  ========= ============
     1    16330   3   2       5262        0         0              194560       37.0        0.0         37.0
     2    17153   1   1      51037        0         0              191290        3.7        0.0          3.7
     3    17447   1   1    1071099       69         0              151101        0.1        2.6          0.1
     4    17218   1   1      59177        0         0              145001        2.5        0.0          2.5
     5    13954   3   4     103267        0         0              142976        1.4        0.0          1.4
     6  2003175   1   5    1426598        0         0              135277        0.1        0.0          0.1
     7  2011920   1   3    1071099        0         0              133735        0.1        0.0          0.1
     8     1201   1   8    1071099        4         2              133628        0.1        5.3          0.1
     9    17154   1   1      32500        0         0              121753        3.7        0.0          3.7
    10    16506   3   3       1510        0         0              100112       66.3        0.0         66.3
    11    16221   3   4     500955        0         0               82858        0.2        0.0          0.2
    12  2002997   1   9       5378        0         0               68060       12.7        0.0         12.7
    13  2010286   1   5     500955        0         0               68005        0.1        0.0          0.1
    14  2010287   1   5     500955        0         0               68005        0.1        0.0          0.1
    15  2010119   1   5     500955        0         0               65958        0.1        0.0          0.1
    16    15997   1   2      93463        0         0               65880        0.7        0.0          0.7
    17  2003176   1   5     601158        0         0               64359        0.1        0.0          0.1
    18     3535   1   8      17656     4276         0               63496        3.6        4.6          3.3
    19  2008666   1   6     500955        0         0               63395        0.1        0.0          0.1
    20    17645   1   1       3809        0         0               59699       15.7        0.0         15.7
    21    13509   1   1     318327        0         0               56772        0.2        0.0          0.2
    22    13855   1   2      43606        0         0               55519        1.3        0.0          1.3
    23    13300   1   2      10619        0         0               49661        4.7        0.0          4.7
    24    13301   1   2      10619        0         0               49661        4.7        0.0          4.7
    25    13932   1   3      43606        0         0               47042        1.1        0.0          1.1
    26  2011066   1   5      38240        0         0               46554        1.2        0.0          1.2
[mpl4@buffalo snort]$
```

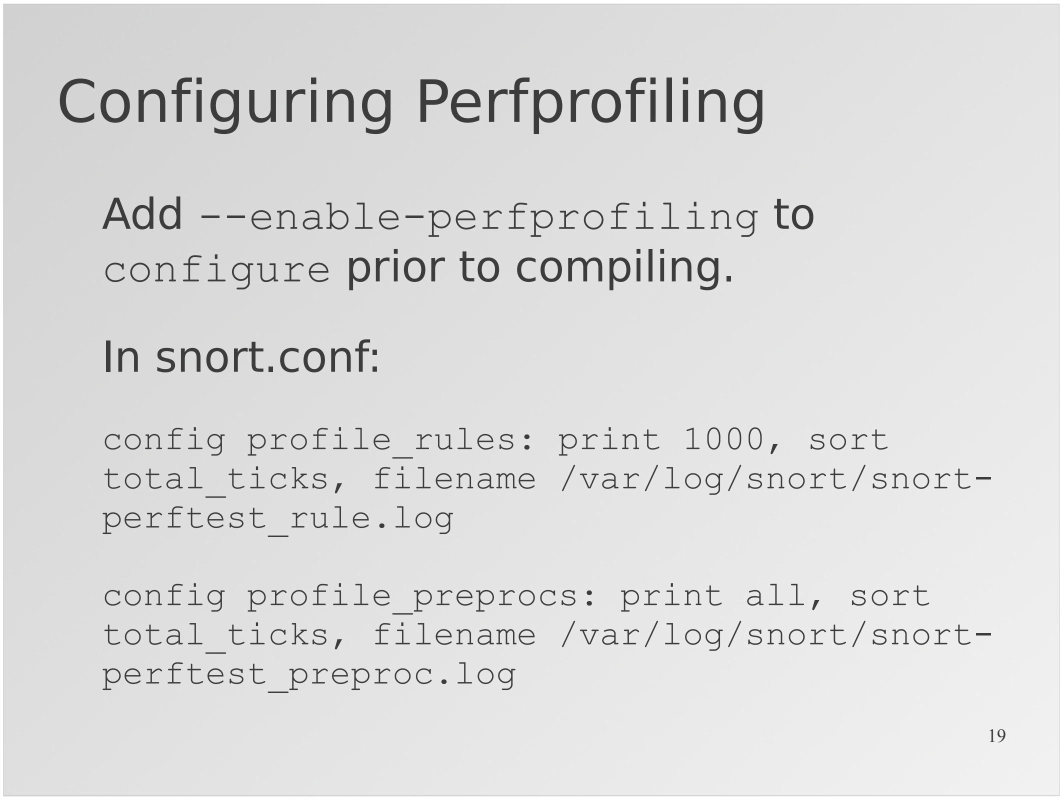
Shows most “expensive” rules in terms of cpu consumption, sorted by microsecs.

A high “check” count indicates a short/common content match causing the rule to evaluate often.

A high “Avg/check” value usually indicates an expensive regex.

Microsecs is checks * avg/check and represents the relative cpu-cost of running the rule.

Rules that “match” a lot but don't “alert” are probably setting flowbits, check to see if you care about that flowbit and if not disable all the rules that set/check it.



## Configuring Perfprofiling

Add `--enable-perfprofiling` to `configure` prior to compiling.

In snort.conf:

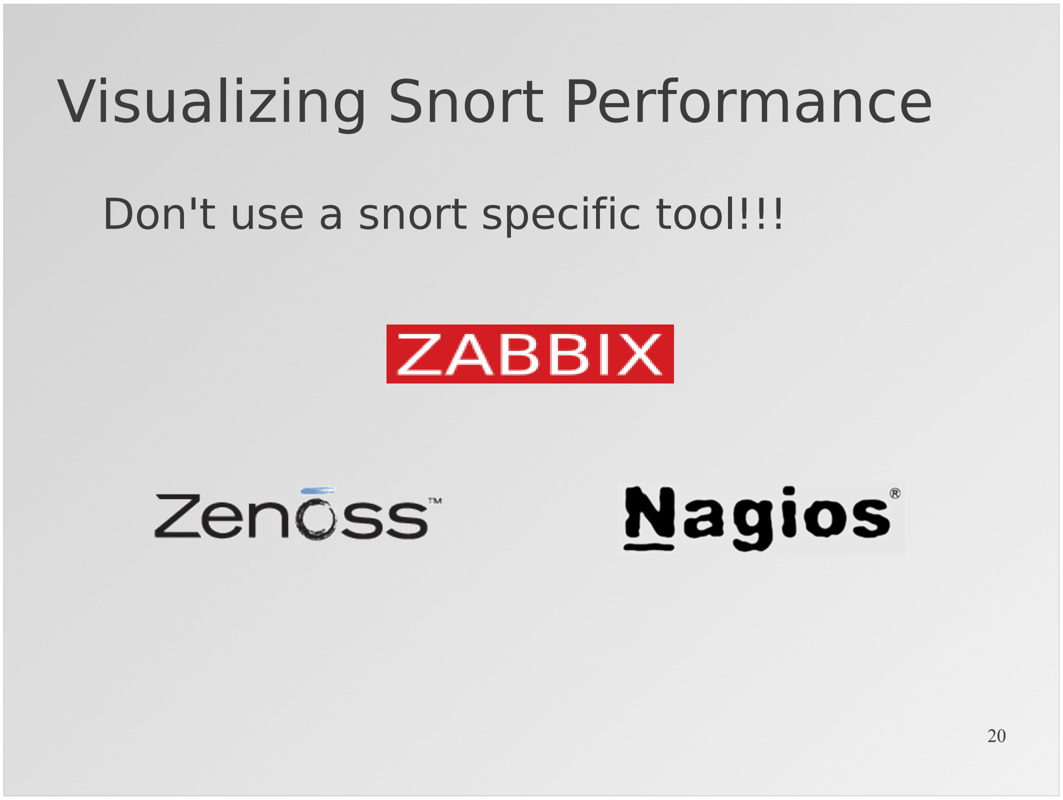
```
config profile_rules: print 1000, sort
total_ticks, filename /var/log/snort/snort-
perftest_rule.log

config profile_preprocs: print all, sort
total_ticks, filename /var/log/snort/snort-
perftest_preproc.log
```

Documented in the perfprofiling readme package with Snort's source:

http://cvs.snort.org/viewcvs.cgi/snort/doc/README.PerfProfiling?rev=HEAD&content-type=text/vnd.viewcvs-markup

There is allegedly a performance overhead for doing this, but it's clearly not all that high. Perhaps 10%-20% at the most, I suspect it's negligible.

Note that this is not trivially collectible or trendable, but it's a good exercise to go through once a year or so during a major upgrade.



## Visualizing Snort Performance

Don't use a snort specific tool!!!

ZABBIX

Zenoss™

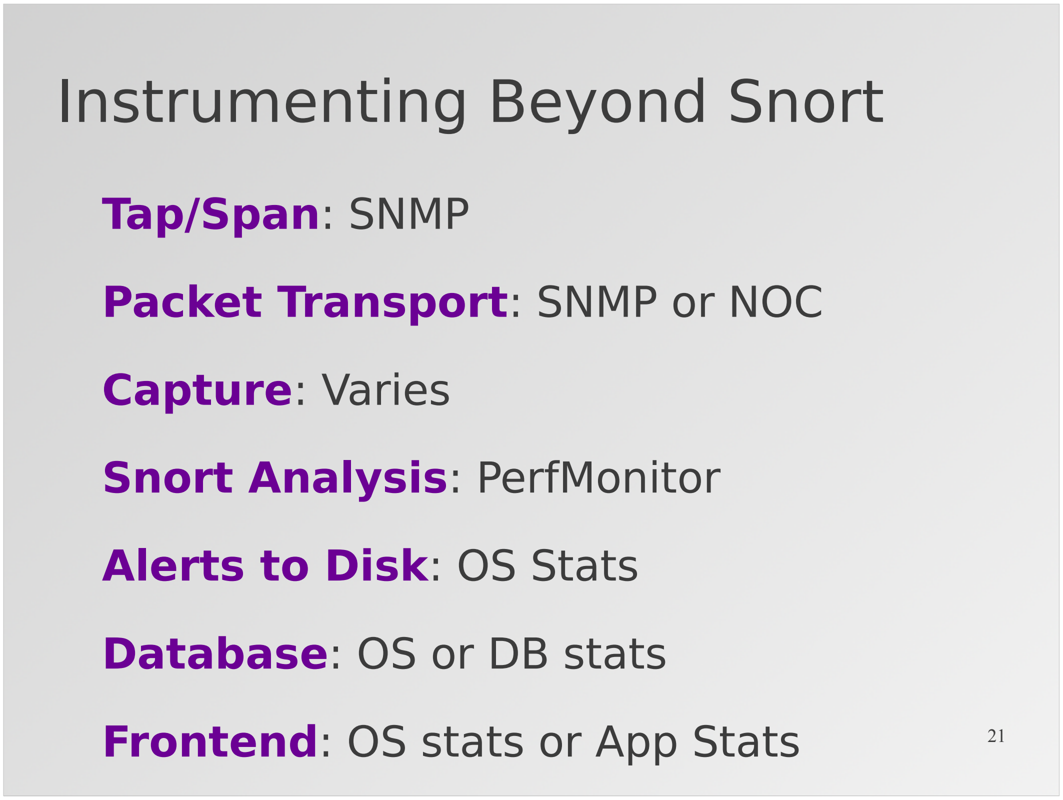
Nagios®

You need data from many sources to evaluate snort performance. Use a tool that can accept data from many sources.

Zabbix has a relatively gentle learning curve and is relatively featureful. Especially dynamically generated graphs are great for exploring data.

I've heard very good things about Zenoss, but haven't used it.

Nagios is well established, but has more legacy baggage and a higher-learning curve. Plus there are no graphing capabilities built in.  Go for it if you have expertise, but I wouldn't stand up a new instance today.



## Instrumenting Beyond Snort

**Tap/Span**: SNMP

**Packet Transport**: SNMP or NOC

**Capture**: Varies

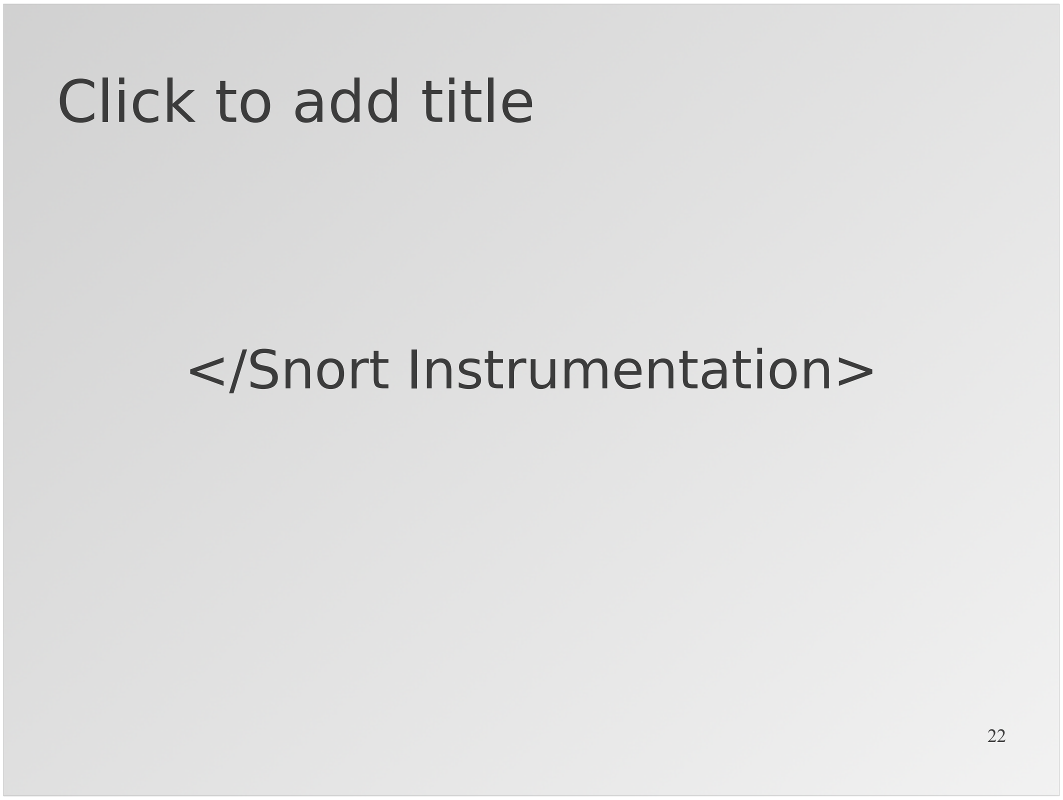
**Snort Analysis**: PerfMonitor

**Alerts to Disk**: OS Stats

**Database**: OS or DB stats

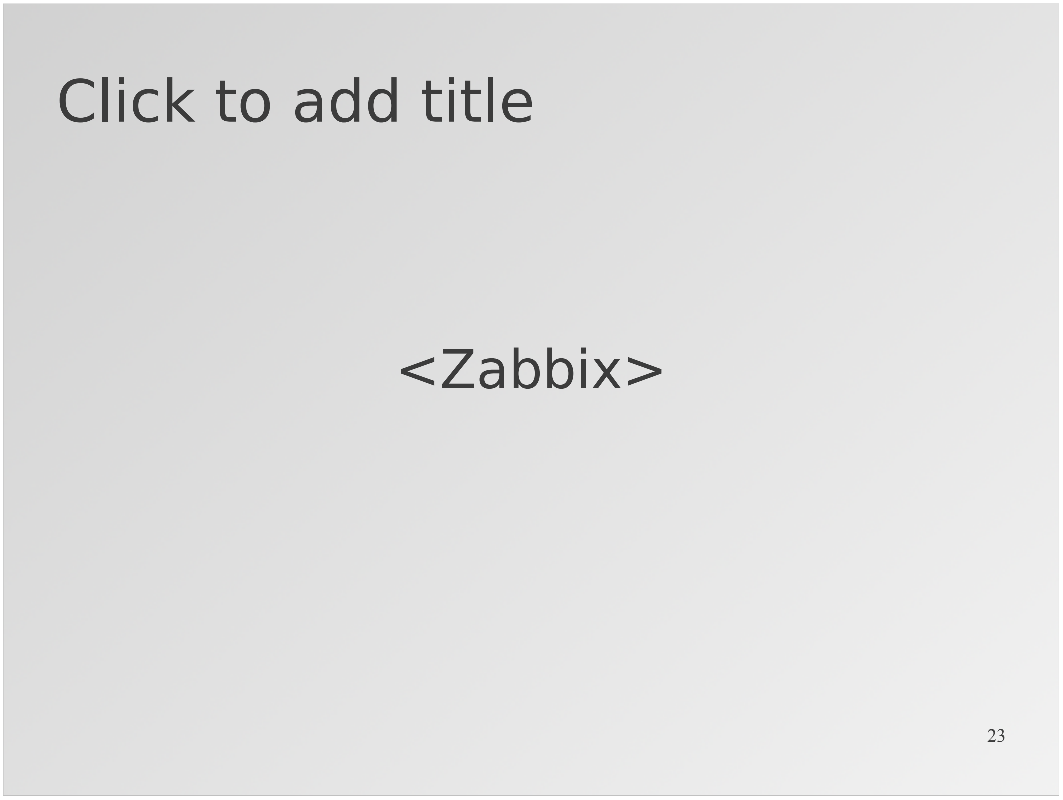
**Frontend**: OS stats or App Stats

Snort is an integrated system that depends on many components, you want a system that can instrument all (or many) of those components, not just snort.



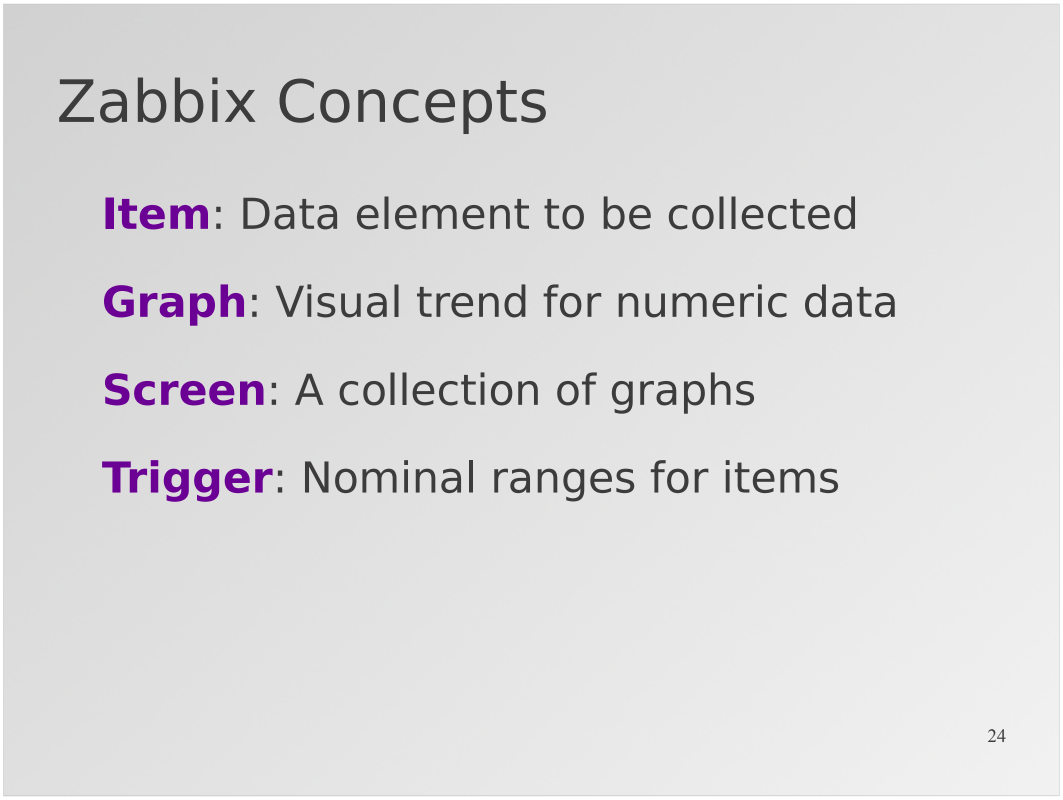
# Click to add title

</Snort Instrumentation>



# Click to add title

<Zabbix>



## Zabbix Concepts

**Item**: Data element to be collected

**Graph**: Visual trend for numeric data

**Screen**: A collection of graphs

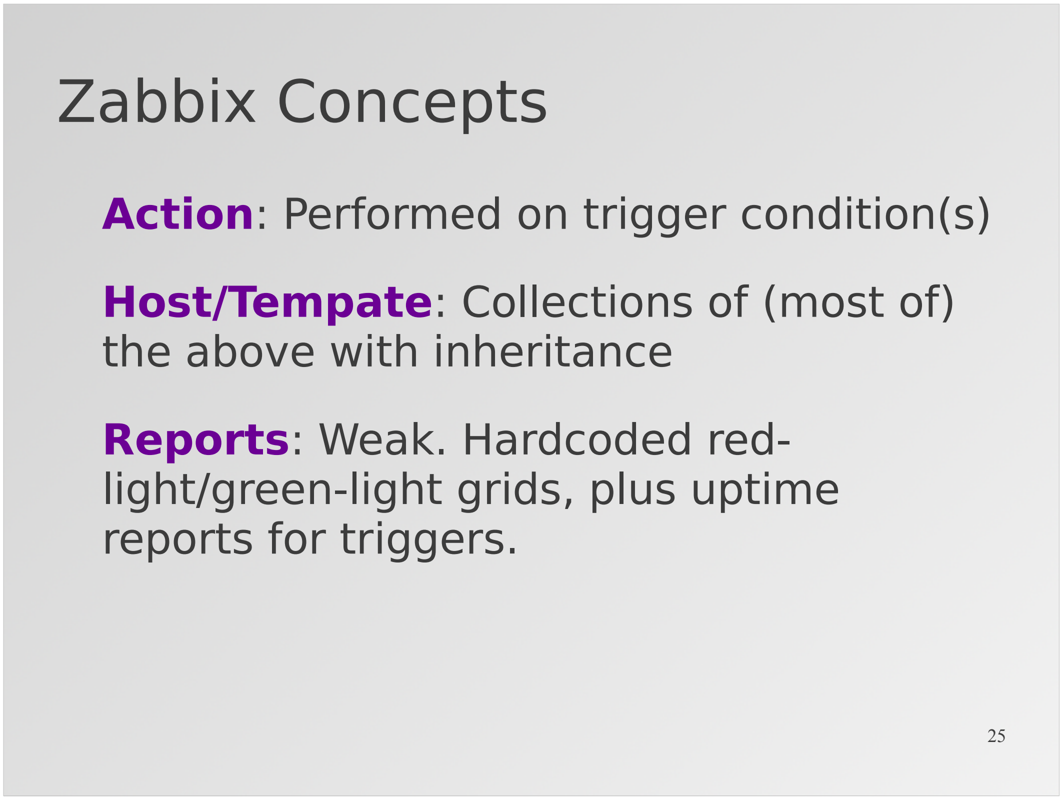
**Trigger**: Nominal ranges for items

Items can be collected from an agent that supports a fairly wide variety of OS items natively. SNMP is also supported, as well as trivial checks like pings or http-requests from the server.

Graphs are generated dynamically at view time, which makes exploration a cinch.

Screens, unlike most other resources can't be templated, which can be frustrating.

Triggers support many conditions including string tests.



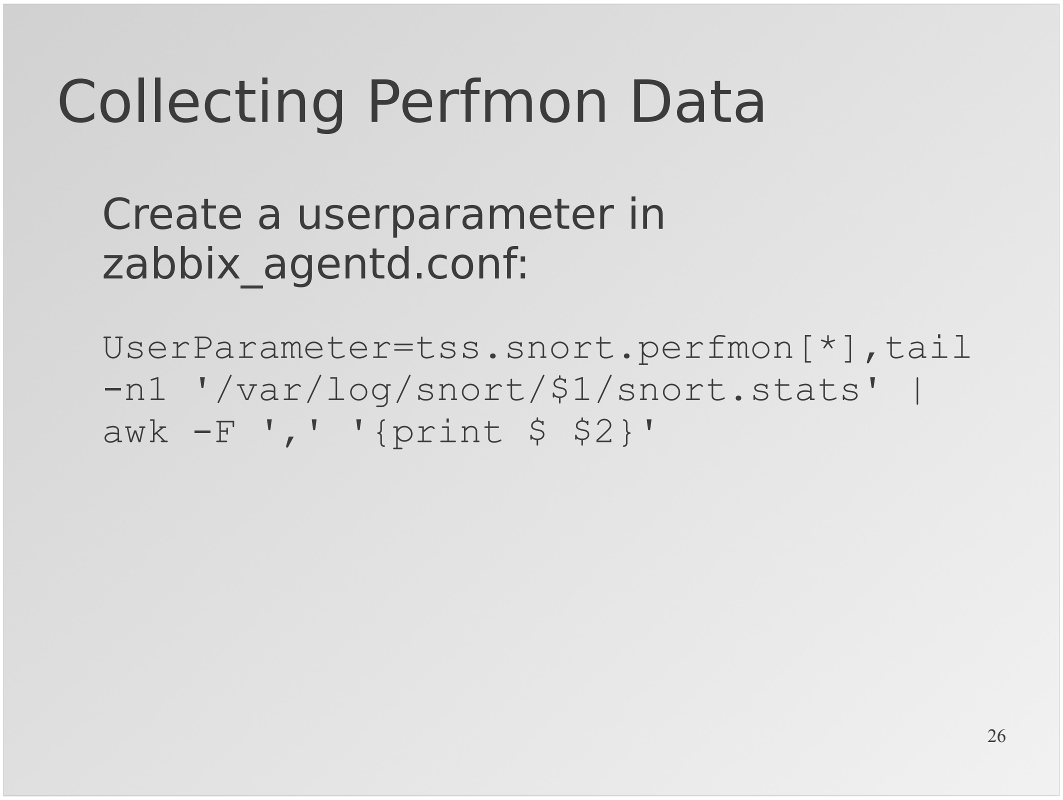
## Zabbix Concepts

**Action**: Performed on trigger condition(s)

**Host/Tempate**: Collections of (most of) the above with inheritance

**Reports**: Weak. Hardcoded red-light/green-light grids, plus uptime reports for triggers.

Notifications are supported via actions, as are IPMI commands (I haven't tested), and remote agent commands (I also haven't tested, but know they require sudo/nopasswd access for the zabbix-agent user). Actions are another resource that can't be templated for no reason that I can fathom.



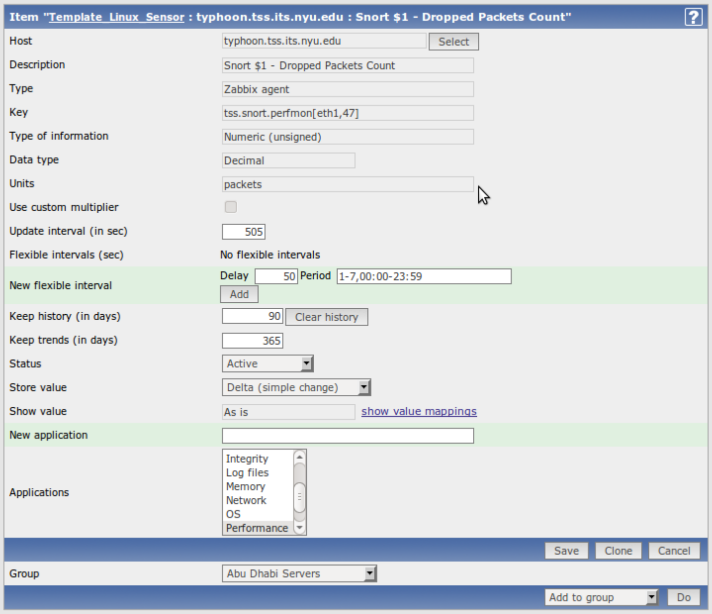
## Collecting Perfmon Data

Create a userparameter in zabbix_agentd.conf:

```
UserParameter=tss.snort.perfmon[*],tail
-n1 '/var/log/snort/$1/snort.stats' |
awk -F ',' '{print $ $2}'
```

This userparameter accepts an interface-name ($1) and an awk column-number ($2). It pulled the most recent line from snort.stats for the appropriate interface, and prints out the column requested (note the extra $, as awk uses them to denote variables just like zabbix does.

# Create an Item



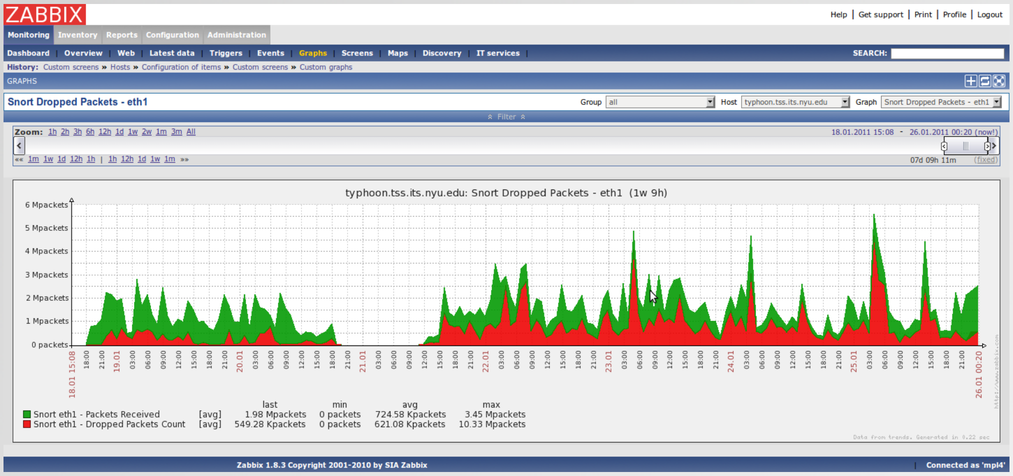
Item "Template_Linux_Sensor : typhoon.tss.its.nyu.edu : Snort $1 - Dropped Packets Count"

| | |
|---|---|
| Host | typhoon.tss.its.nyu.edu |
| Description | Snort $1 - Dropped Packets Count |
| Type | Zabbix agent |
| Key | tss.snort.perfmon[eth1,47] |
| Type of information | Numeric (unsigned) |
| Data type | Decimal |
| Units | packets |
| Use custom multiplier | □ |
| Update interval (in sec) | 505 |
| Flexible intervals (sec) | No flexible intervals |
| New flexible interval | Delay 50 Period 1-7,00:00-23:59 |
| Keep history (in days) | 90 |
| Keep trends (in days) | 365 |
| Status | Active |
| Store value | Delta (simple change) |
| Show value | As is |
| New application | |
| Applications | Integrity<br>Log files<br>Memory<br>Network<br>OS<br>Performance |
| Group | Abu Dhabi Servers |

This is my dropped packets item.

Note “Store Value” → Delta setting. The snort dropped packet stats are poor in my opinion due to averaging over the lifetime of the process, and this stat is given as a simple counter. Zabbix converts the counter into a number of dropped packets per collection-period and gracefully handles overflow-rollovers.

# Create a Graph


ZABBIX
Help | Get support | Print | Profile | Logout
Monitoring Inventory Reports Configuration Administration
Dashboard | Overview | Web | Latest data | Triggers | Events | Graphs | Screens | Maps | Discovery | IT services |
SEARCH:
GRAPHS
Filter
Zoom: 1h 2h 3h 6h 12h 1d 1w 2w 1m 3m All
«« 1m 1w 1d 12h 1h | 1h 12h 1d 1w 1m »»
(fixed)
6 Mpackets
5 Mpackets
4 Mpackets
3 Mpackets
2 Mpackets
1 Mpackets
0 packets
18.01 15:08
19.01
20.01
21.01
22.01
23.01
24.01
25.01
26.01 00:20

History: Custom screens » Hosts » Configuration of items » Custom screens » Custom graphs

Snort Dropped Packets - eth1

Group: all   Host: typhoon.tss.its.nyu.edu   Graph: Snort Dropped Packets - eth1

18.01.2011 15:08 - 26.01.2011 00:20 (now!)

07d 09h 11m

typhoon.tss.its.nyu.edu: Snort Dropped Packets - eth1 (1w 9h)

| | | last | min | avg | max |
|---|---|---|---|---|---|
| 🟩 Snort eth1 - Packets Received | [avg] | 1.98 Mpackets | 0 packets | 724.58 Kpackets | 3.45 Mpackets |
| 🟥 Snort eth1 - Dropped Packets Count | [avg] | 549.28 Kpackets | 0 packets | 621.08 Kpackets | 10.33 Mpackets |

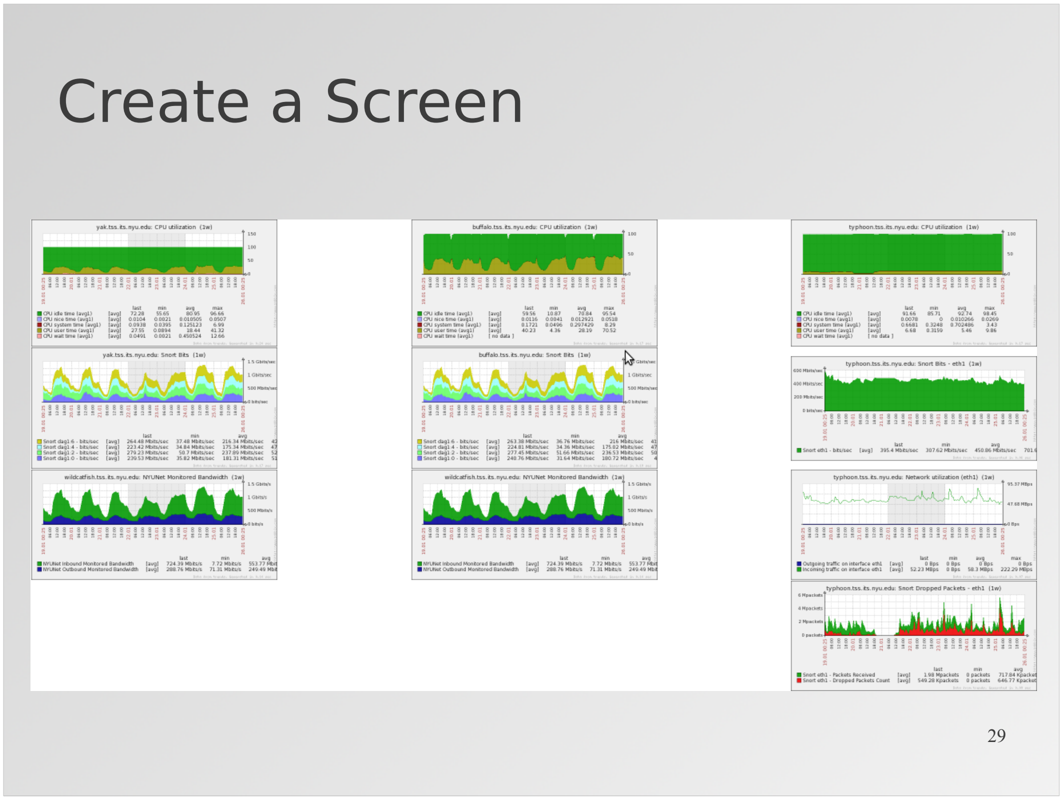
Zabbix 1.8.3 Copyright 2001-2010 by SIA Zabbix

Connected as 'mpl4'

Line graphs or pie graphs.

Line graphs can be stacked, as above, to show cumulative values over time. This snort sensor is angry, red is dropped packets, green is processed packets.

# Create a Screen

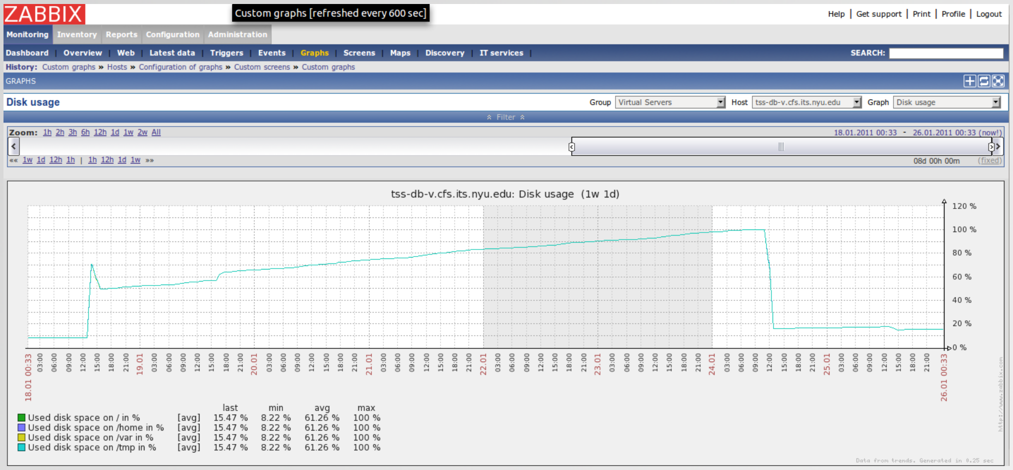


Three sensors, in columns.

In rows from top to bottom are: cpu-utilization (normalized against the number of cpu's present), bandwidth reported by snort (stacked where there are multiple snort instances), bandwidth reported by the gigamon (or kernel on the right), packet processed vs dropped packets where available.

This kind of at-a-glance analysis really changed the way we operate.

# Find Problems – DB Disk Outage

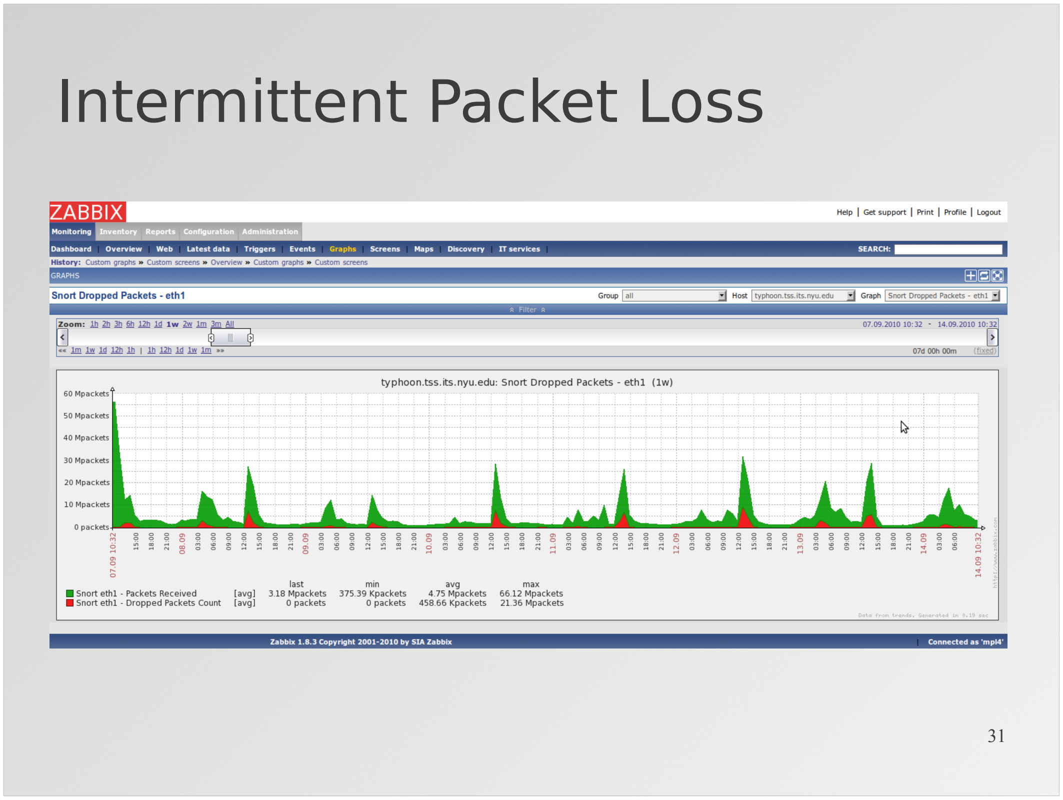

ZABBIX
Custom graphs [refreshed every 600 sec]
Help | Get support | Print | Profile | Logout
Monitoring Inventory Reports Configuration Administration
Dashboard | Overview | Web | Latest data | Triggers | Events | Graphs | Screens | Maps | Discovery | IT services |
SEARCH:
History: Custom graphs » Hosts » Configuration of graphs » Custom screens » Custom graphs
GRAPHS
Disk usage
Group Virtual Servers
Host tss-db-v.cfs.its.nyu.edu
Graph Disk usage
Filter
Zoom: 1h 2h 3h 6h 12h 1d 1w 2w All
18.01.2011 00:33 - 26.01.2011 00:33 (now!)
«« 1w 1d 12h 1h | 1h 12h 1d 1w »»
08d 00h 00m (fixed)
tss-db-v.cfs.its.nyu.edu: Disk usage (1w 1d)
120 %
100 %
80 %
60 %
40 %
20 %
0 %
18.01 00:33
19.01
20.01
21.01
22.01
23.01
24.01
25.01
26.01 00:33
03:00
06:00
09:00
12:00
15:00
18:00
21:00
http://www.zabbix.com
Data from trends. Generated in 0.25 sec

| | | last | min | avg | max |
|---|---|---|---|---|---|
| 🟩 Used disk space on / in % | [avg] | 15.47 % | 8.22 % | 61.26 % | 100 % |
| 🟦 Used disk space on /home in % | [avg] | 15.47 % | 8.22 % | 61.26 % | 100 % |
| 🟨 Used disk space on /var in % | [avg] | 15.47 % | 8.22 % | 61.26 % | 100 % |
| 🟦 Used disk space on /tmp in % | [avg] | 15.47 % | 8.22 % | 61.26 % | 100 % |

This was a real snort outage, and this is the real graph that I found the problem with.

This was a disk-space outage on our DB server (the scale goes to 120%, but my disks do not), but I actually had to investigate my whole failure chain to find it. Doing so in zabbix took only a few minutes.

Proper use of triggers and actions/alerts could have prevented this from happening at all.



# Intermittent Packet Loss

History: Custom graphs » Custom screens » Overview » Custom graphs » Custom screens

**Snort Dropped Packets - eth1**

Group: all
Host: typhoon.tss.its.nyu.edu
Graph: Snort Dropped Packets - eth1

07.09.2010 10:32 - 14.09.2010 10:32

07d 00h 00m

typhoon.tss.its.nyu.edu: Snort Dropped Packets - eth1 (1w)

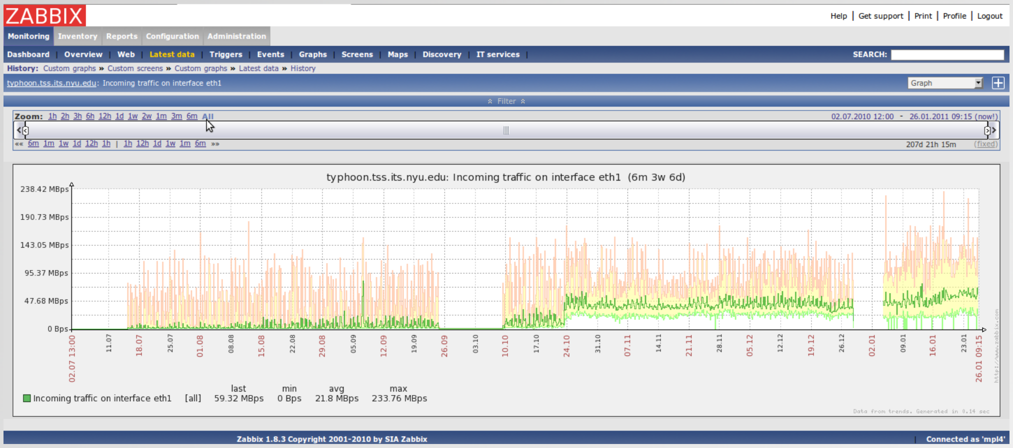
| | | last | min | avg | max |
|---|---|---|---|---|---|
| 🟩 Snort eth1 - Packets Received | [avg] | 3.18 Mpackets | 375.39 Kpackets | 4.75 Mpackets | 66.12 Mpackets |
| 🟥 Snort eth1 - Dropped Packets Count | [avg] | 0 packets | 0 packets | 458.66 Kpackets | 21.36 Mpackets |

Data from trends. Generated in 0.19 sec

Zabbix 1.8.3 Copyright 2001-2010 by SIA Zabbix | Connected as 'mpl4'

This sensor can achieve acceptable loss stats by filtering one port on one host that does large nightly transfers.

# Traffic Growth



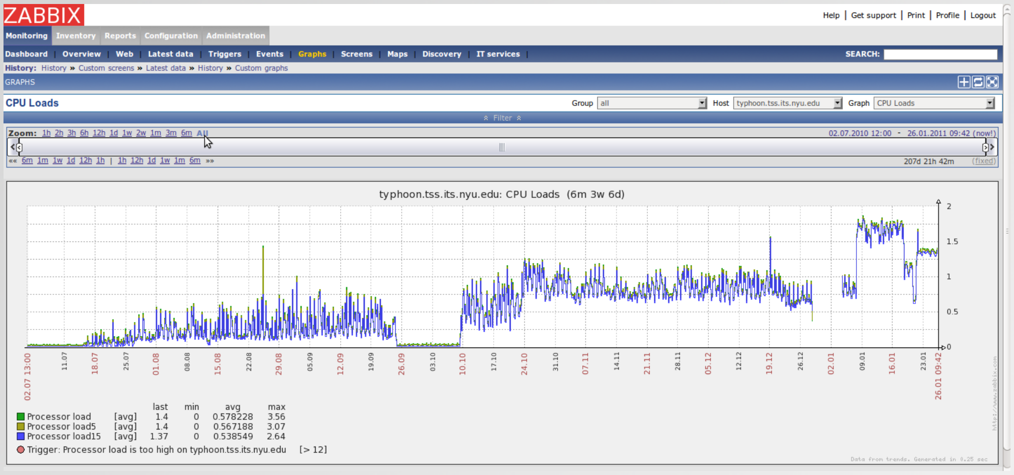
ZABBIX
Help | Get support | Print | Profile | Logout
Monitoring Inventory Reports Configuration Administration
Dashboard | Overview | Web | Latest data | Triggers | Events | Graphs | Screens | Maps | Discovery | IT services |
SEARCH:
History: Custom graphs » Custom screens » Custom graphs » Latest data » History
typhoon.tss.its.nyu.edu: Incoming traffic on interface eth1
Graph
Filter
Zoom: 1h 2h 3h 6h 12h 1d 1w 2w 1m 3m 6m All
02.07.2010 12:00 - 26.01.2011 09:15 (now!)
«« 6m 1m 1w 1d 12h 1h | 1h 12h 1d 1w 1m 6m »»
207d 21h 15m (fixed)
typhoon.tss.its.nyu.edu: Incoming traffic on interface eth1 (6m 3w 6d)
238.42 MBps
190.73 MBps
143.05 MBps
95.37 MBps
47.68 MBps
0 Bps
02.07 13:00
11.07
18.07
25.07
01.08
08.08
15.08
22.08
29.08
05.09
12.09
19.09
26.09
03.10
10.10
17.10
24.10
31.10
07.11
14.11
21.11
28.11
05.12
12.12
19.12
26.12
02.01
09.01
16.01
23.01
26.01 09:15
Data from trends. Generated in 0.14 sec
Zabbix 1.8.3 Copyright 2001-2010 by SIA Zabbix
Connected as 'mpl4'

| | | last | min | avg | max |
|---|---|---|---|---|---|
| 🟩 Incoming traffic on interface eth1 | [all] | 59.32 MBps | 0 Bps | 21.8 MBps | 233.76 MBps |

This sensor was spec'ed for 50-100mbits of traffic. The two previous dropped-packets graphs are for this host at different times during it's traffic growth to now 500mbit/sec.

Take note of the gaps for comparison on the next slide.

# Span Outage

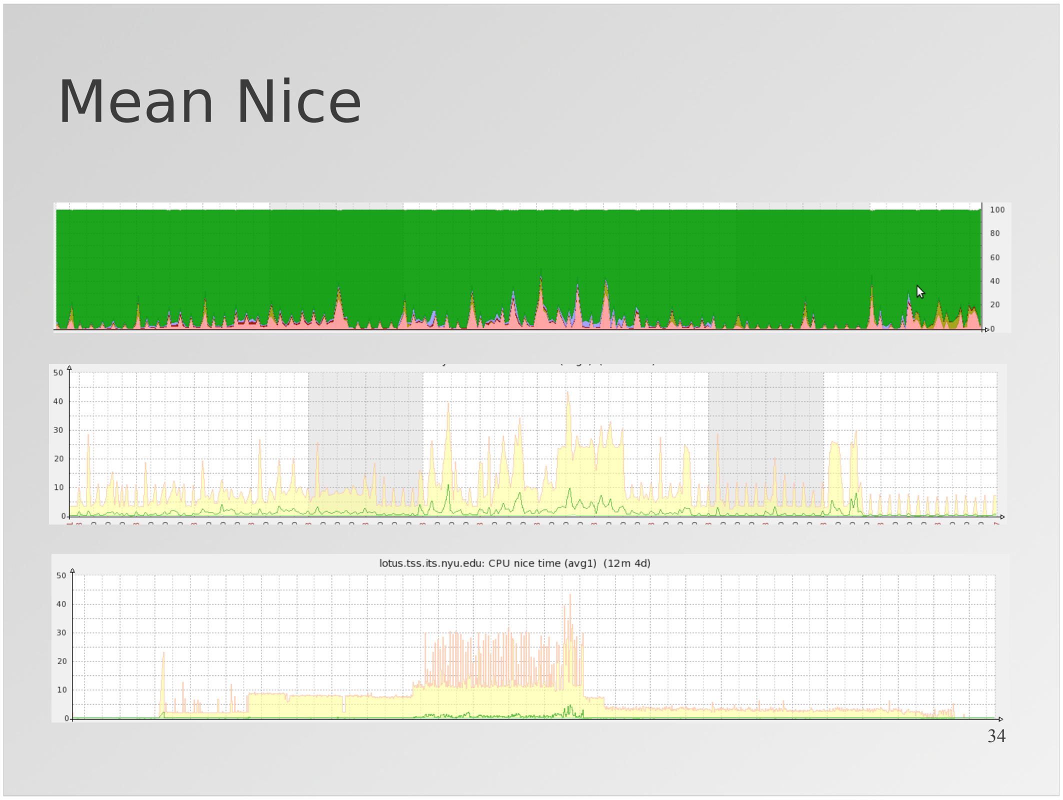

ZABBIX
Help | Get support | Print | Profile | Logout
Monitoring Inventory Reports Configuration Administration
Dashboard | Overview | Web | Latest data | Triggers | Events | Graphs | Screens | Maps | Discovery | IT services |
SEARCH:
History: History » Custom screens » Latest data » History » Custom graphs
GRAPHS
CPU Loads
Group all
Host typhoon.tss.its.nyu.edu
Graph CPU Loads
« Filter «
Zoom: 1h 2h 3h 6h 12h 1d 1w 2w 1m 3m 6m All
02.07.2010 12:00 - 26.01.2011 09:42 (now!)
«« 6m 1m 1w 1d 12h 1h | 1h 12h 1d 1w 1m 6m »»
207d 21h 42m
(fixed)
typhoon.tss.its.nyu.edu: CPU Loads (6m 3w 6d)
2
1.5
1
0.5
0
02.07 13:00
11.07
18.07
25.07
01.08
08.08
15.08
22.08
29.08
05.09
12.09
19.09
26.09
03.10
10.10
17.10
24.10
31.10
07.11
14.11
21.11
28.11
05.12
12.12
19.12
26.12
02.01
09.01
16.01
23.01
26.01 09:42
http://www.zabbix.com
Data from trends. Generated in 0.25 sec

| | | last | min | avg | max |
|---|---|---|---|---|---|
| 🟩 Processor load | [avg] | 1.4 | 0 | 0.578228 | 3.56 |
| 🟩 Processor load5 | [avg] | 1.4 | 0 | 0.567188 | 3.07 |
| 🟦 Processor load15 | [avg] | 1.37 | 0 | 0.538549 | 2.64 |

● Trigger: Processor load is too high on typhoon.tss.its.nyu.edu [> 12]

The second gap also has a gap in the system load graph, this was a planned outage.

The first gap has low system load stats. Our span was misconfigured by the NOC and we stopped getting packets. Zabbix told us before our incident monitoring folks realized that alerts had stopped.

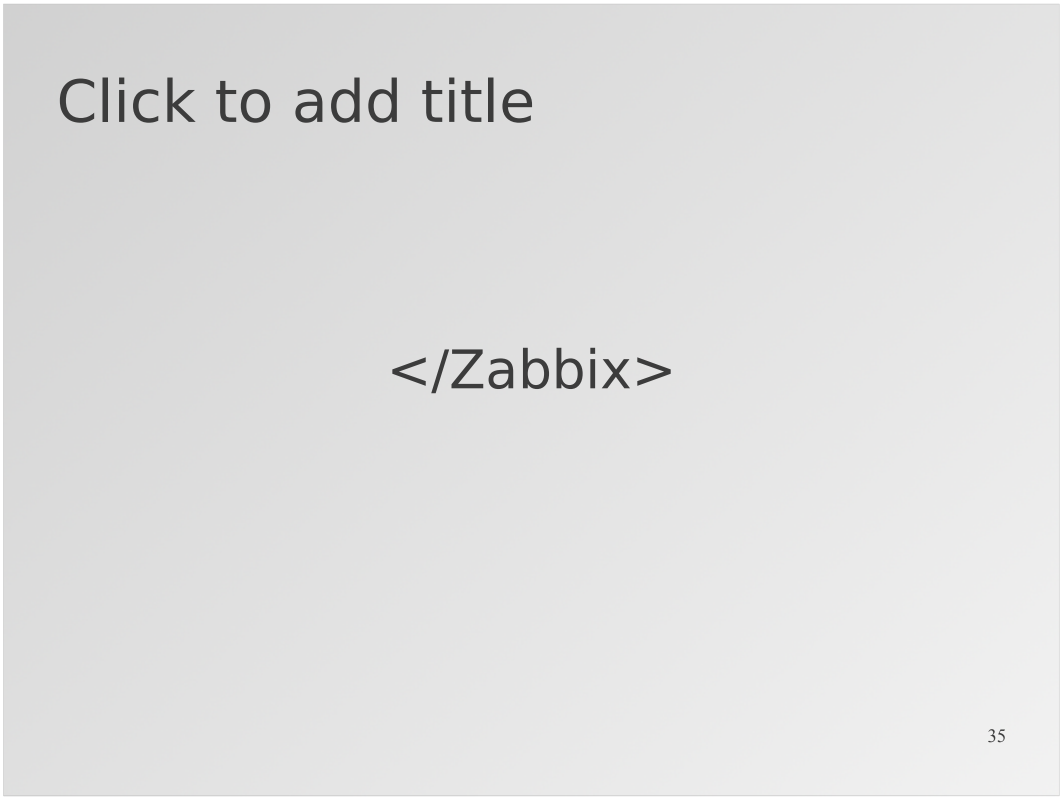


# Mean Nice

This was a bear to troubleshoot. Processes were getting reniced and we couldn't figure out how.

Turns out it was updated processes inheriting the nice-value of yum-updatesd when being restarted after upgrades. We finally made this determination by correlating log timestamps with initial nice-cpu spikes. We couldn't have done it based on logs alone, we needed the CPU trend data.

This is an outstanding redhat bug by the way, watch out for it.



# Click to add title

</Zabbix>

# Click to add title

<Demo />

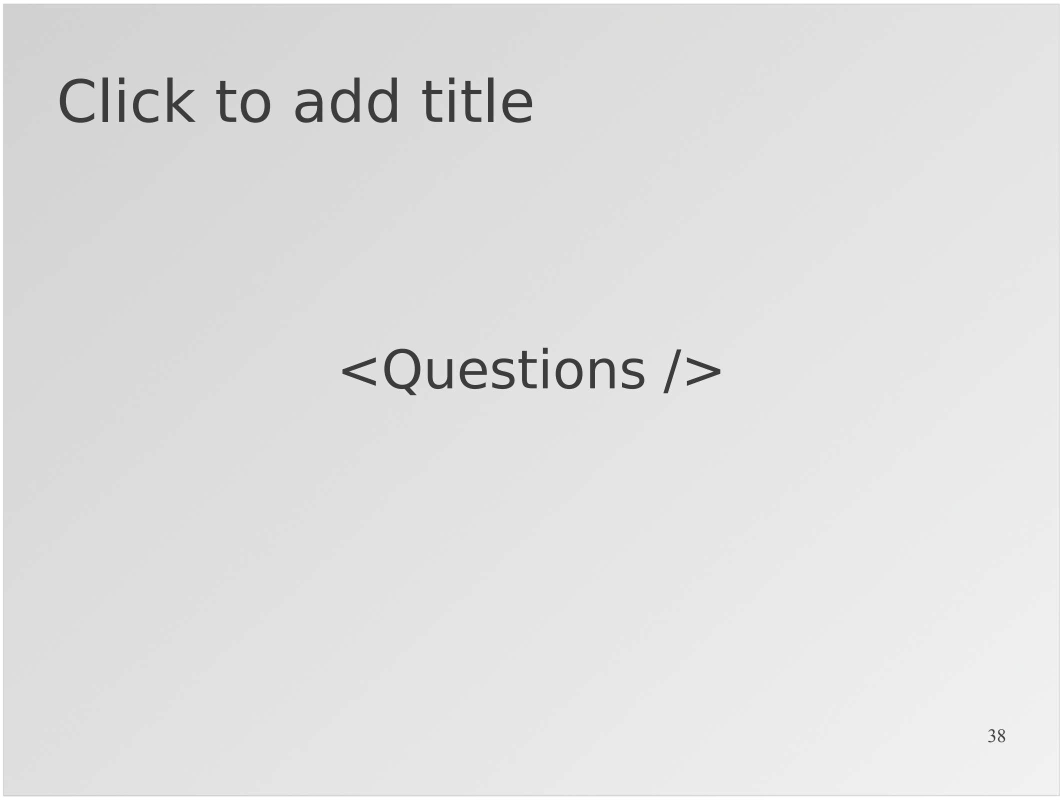
# Conclusion

**Instrument** your systems

**Visualize** your data

**Troubleshoot** faster

**Understand** what's normal for you



# Click to add title

<Questions />