Using Perfmon and Performance Profiling to Tune Snort Preprocessors and Rules

06 November 2009

Prepared By:
Steven Sturges

Sourcefire, Incorporated
9770 Patuxent Woods Drive
Columbia, MD 21046

Overview

This document describes guidelines for tuning Snort based on performance statistics from the rule and preprocessor profiling and perfmon preprocessor.

It is intended for Snort administrators, to be used when tuning and troubleshooting performance problems.

Perfmon Preprocessor

The perfmon preprocessor has been included with Snort since Snort 1.9. There are a number of aspects of the perfmon data that help a user tune Snorts configuration.

Some of the individual data items listed in the Snort manual under the Performance Monitor Preprocessor section will be used. Understanding the meaning of those performance metrics is paramount to tuning Snort.

A sample Perfmon Preprocessor Output from console is shown below from a test pcap in readback mode. With pcap readback mode, there will be no dropped packets.

```
Snort Realtime Performance  : Wed Aug 19 10:23:07 2009
---------------------------
Pkts Recv:   1858011
Pkts Drop:   0
% Dropped:   0.000%
Blocked:     0
Pkts Filtered TCP:     0
Pkts Filtered UDP:     0

Mbits/Sec:   139.663 (wire)
Mbits/Sec:   0.000 (mpls)
Mbits/Sec:   0.391 (ip fragmented)
Mbits/Sec:   0.377 (ip reassembled)
Mbits/Sec:   3.162 (tcp rebuilt)
Mbits/Sec:   142.434 (app layer)

Bytes/Pkt:   609 (wire)
Bytes/Pkt:   0 (mpls)
Bytes/Pkt:   825 (ip fragmented)
Bytes/Pkt:   1692 (ip reassembled)
Bytes/Pkt:   1024 (tcp rebuilt)
Bytes/Pkt:   613 (app layer)

KPkts/Sec:   28.620 (wire)
KPkts/Sec:   0.000 (mpls)
KPkts/Sec:   0.059 (ip fragmented)
KPkts/Sec:   0.028 (ip reassembled)
KPkts/Sec:   0.386 (tcp rebuilt)
KPkts/Sec:   29.033 (app layer)

PatMatch:    23.521%

CPU Usage:   89.562% (user)  3.581% (sys)  6.857% (idle)
```

```
Alerts/Sec             :  53.927
Syns/Sec               :  1313.853
Syn-Acks/Sec           :  269.068
New Cached Sessions/Sec:  757.819
Midstream Sessions/Sec :  64.617
Cached Sessions Del/Sec:  757.819
Closed Sessions/Sec    :  116.958
TimedOut Sessions/Sec  :  627.367
Pruned Sessions/Sec    :  112.122
Dropped Async Ssns/Sec :  0.000
Current Cached Sessions:  0
Sessions Initializing  :  34
Sessions Established   :  33
Sessions Closing       :  0
Max Cached Sessions    :  8179
Max Sessions (interval):  8179
Stream Flushes/Sec     :  385.733
Stream Cache Faults/Sec:  0
Stream Cache Timeouts  :  40729
Frag Creates()s/Sec    :  29.575
Frag Completes()s/Sec  :  27.865
Frag Inserts()s/Sec    :  29.559
Frag Deletes/Sec       :  29.575
Frag AutoFrees/Sec     :  0.000
Frag Flushes/Sec       :  27.865
Current Cached Frags   :  0
Max Cached Frags       :  113
Frag Timeouts          :  0
Frag Faults            :  0

New Cached UDP Ssns/Sec:  0.000
Cached UDP Ssns Del/Sec:  0.000
Current Cached UDP Ssns:  0
Max Cached UDP Ssns    :  0

Attribute Table Hosts  :  0
Attribute Table Reloads:  0
```

Throughputs and Other Rates

The first group of metrics reported by perfmon is rates for various aspects of Snort's detection. Those include a drop rate, as reported by pcap_stats(), megabits per second (processed by Snort), alerts per second, packets per second and bytes per packet. The better Snort is performing, the lower the drop rate, higher the megabits per second (throughput), higher the average packet size, etc.

Pattern Matching

The next statistic is pattern match percentage. This is the number of bytes that Snort is passing through the pattern matcher to identify possible rules, compared to the total number of bytes seen by Snort. This number could be higher than 100%, in the case of IP defragmentation, TCP reassembly, DCE/RPC reassembly, etc. Ideally this would be in the 10% range.

- To reduce the pattern match percentage, it is first best to eliminate un-necessary reassembly (only reassembly on the ports and services that are critical to protect – and only one side of those connections. This can be done through proper configuration of Stream TCP reassembly.
- If possible, eliminate traffic from being inspected by Snort at all via an `ignore_ports` configuration option as well as the use of various preprocessors (SSL, SSH, FTP/Telnet, and others) to not inspect encrypted traffic and file transfers. Since native Snort cannot decrypt traffic, inspecting encrypted traffic over SSL or SSH is a waste of system resources. Additionally, a BPF can be specified to cause Snort to completely ignore the traffic matching the filter criteria.
- Reduce the amount of HTTP client and server traffic that is inspected by using HTTP Inspect's `client_flow_depth` and `server_flow_depth` options.
- Reduce the amount of DCE/RPC reassembled data that is inspected by using the DCE/RPC `max_frag_len` option.
- One of the features of TCP reassembly is to eliminate evasions of items split across packets. If there are many large packets on the network, use the Stream5 TCP Option `dont_store_large_packets` to eliminate the inclusion of those large packets in TCP reassembly.

Amount of total processing time for pattern matching will also be affected by these settings. See section 0 for details on preprocessor profiling and processing time.

Stream Performance

The next group of statistics reported by perfmon relates to TCP Stream (Stream5) and includes rates for creating new connections, terminating connections, and TCP session cache performance. A few key statistics here related to each other include the rate of new and deleted sessions, current sessions in the cache, maximum number of sessions in the cache, cache faults and timeouts. Further breakdowns of the TCP stats in terms of rates for sessions that are created mid-stream, closed normally, pruned for memory constraints, or timed-out are also available.

- Large numbers of cache faults could result in thrashing of the TCP session cache and can be addressed by proper configuration of the Stream `memcap` and `max_tcp` settings. If the `memcap` is too low relative to the `max_tcp` and the TCP ports used for reassembly, memory starvation will occur. Snort will self-regulate by deleting older (least recently used) TCP sessions from the cache even though those sessions may still be active on the wire.
- Snort is best not deployed in an asymmetric traffic environment. Doing so will result in large numbers of stream cache faults and/or timeouts. An easy way to identify this is to compare the SYNs per second to the SYN/ACKs per second. A greater than 2-1 ratio could indicate a routing or retransmit issue on your network.
- A low setting for the Stream `timeout` configuration will cause more sessions to timeout abnormally, especially if the applications and/or firewall controlling the traffic being seen by Snort use significantly different timeouts.

- Whether from a misconfiguration of the Stream `memcap`, `max_tcp`, or `timeout`, when traffic from a still active session that was deleted from the cache is seen by Snort, Snort will delete another older session. If poorly configured, this will cause a thrashing of deleting and re-creating TCP sessions. This leads to both poor performance and evasion opportunities.
- Recommended settings for a system w/ 2GB of system memory, assuming 1GB available to Snort, are 256k `max_tcp` and a `memcap` of 137MB. This is assuming the default set of ports are used for TCP reassembly and a traffic profile of 15% DCE/SMB, 15% SMTP, 10% DNS, 10% FTP, and the rest HTTP.
- Similar to TCP sessions, the UDP session cache has stats that reflect its performance. Recommended values for a similar system are 64k for `max_udp`. That should be configured relative to the amount of UDP traffic on the network being monitored.

IP Defrag Performance

As is the case with TCP Stream reassembly, improperly configured IP defragmentation (frag3) settings will adversely affect performance as well as lead to evasion opportunities. If IP defragmentation is not seen on your network, simply turn off frag3, to eliminate a small performance hit when Snort checks if packets are fragmented.

If the network does experience fragmentation, a number of statistics reported by perfmon are available to help tune Frag3. In particular, the Frag-Auto Deletes, Frag-Flushes, Frag-Current, Frag-Max, Frag-Timeouts, and Frag-Faults values are important. Frag Auto-Deletes and Frag-Faults relate to the memory settings whether preallocating fragment storage or using a memcap.

The Frag3 `max_frags` limits the number of simultaneous in process defragmentations, while the `memcap` limits the memory used to store individual fragment nodes.

Preallocation of fragment nodes will help performance because Snort will not incur an expensive memory allocation and free when fragments are stored and later used, but rather allocate at initialization and recycle the memory. Use either `prealloc_memcap` or `prealloc_frags` to preallocate fragment nodes. The size of the fragment node is based on the snap length, so adjust these values as the frame size changes.

CPU Usages

When performance statistics are generated Snort will capture the CPU usage on the system as reported by the kernel or proc filesystem. The CPU usage is a snapshot, so for a single instance of performance statistics, it is not indicative of an issue. However, consistent high CPU usage, where the user and/or system approach 100% and idle is near 0% is indicative that Snort or some other process is consuming most of the CPU.

Tuning Snort's preprocessors and the rule set as described throughout this document will help alleviate high CPU consumption.

Averages

The next group of statistics is a breakdown of averages of throughput, bytes per packet, and packets per second for 5 areas: wire, ip fragmented, ip defragmented, tcp reassembled, and application layer. The numbers for the application layer are the most important, as those are the ones bytes/packets that are passed through the application preprocessors for normalization and rule inspection. Reducing the application values through the means noted above in section 0 – BPFs, `ignore_ports`, and/or ignoring encrypted connections – will help improve the performance of Snort for the traffic that is important to inspect.

Snort also logs stats relating to the number of packets pcap received, number of packets pcap dropped (that didn't reach Snort because of high CPU usage), number blocked (when inline), and a drop percentage. Ideally, Snort will be tuned such that the drop percentage is 0.

Using Flow Data

Perfmon has an option that prints out stats about packet sizes and flow data. A sample of that is shown below from the same sample pcap. This information may be useful in tuning both the application preprocessors and rules.

The first category (Protocol Byte Flows) shows a breakdown of how much data is sent across each protocol that Snort inspects.

```
Protocol Byte Flows - %Total Flow
-------------------------------------
TCP:   95.60%
UDP:   3.69%
ICMP:  0.15%
OTHER: 0.57%
```

The next category (PacketLen) shows the distribution of the packet sizes, independent of protocol. TCP ACK packets contribute significantly to the smaller packet sizes (60-66 bytes). If a significant percentage of traffic is large byte TCP packets, Stream5 TCP `dont_store_large_packets` option may be used to eliminate it from inclusion in TCP reassembled packets. In the example data, this would apply if the many of the 30+% of 1514 byte packets are TCP and would be included in TCP reassembly.

```
PacketLen - %TotalPackets
-------------------------
Bytes[60] 22.72%
Bytes[62] 1.70%
Bytes[66] 12.56%
Bytes[70] 0.55%
Bytes[74] 5.23%
Bytes[78] 1.05%
Bytes[80] 0.11%
Bytes[86] 0.24%
Bytes[87] 0.12%
Bytes[91] 0.10%
Bytes[92] 0.58%
Bytes[98] 1.72%
Bytes[102] 0.18%
```

```
Bytes[106] 0.10%
Bytes[110] 0.69%
Bytes[114] 2.87%
Bytes[118] 0.64%
Bytes[126] 0.16%
Bytes[130] 0.27%
Bytes[134] 0.19%
Bytes[146] 0.66%
Bytes[162] 0.27%
Bytes[166] 0.16%
Bytes[342] 0.78%
Bytes[566] 0.64%
Bytes[590] 1.09%
Bytes[1314] 0.97%
Bytes[1374] 0.20%
Bytes[1434] 0.96%
Bytes[1460] 0.10%
Bytes[1474] 0.28%
Bytes[1486] 0.13%
Bytes[1514] 30.52%
```

The categories that follow give the distribution of the ports across the different protocols.

The per protocol port distribution is important to know because that will allow proper configuration of the preprocessors. For example, the pcap has almost 10% traffic across TCP port 22 (SSH). If the SSH preprocessor is not in use in the configuration, that is traffic that Snort is needlessly inspecting. Similarly, with traffic on port 443 (HTTPS) and 993 (IMAPS), the SSL preprocessor should be turned on to ignore that traffic.

On the UDP side, there is traffic on port 137 and 138, so if the rule set doesn't include Netbios Datagram or Nameservice rules, those can be ignored via the `ignore_ports` option or a BPF. Similarly, traffic on ports 67 & 68 is the BOOTP service or DHCP, so this traffic could be ignored if there are no DHCP rules in the rule set.

If there are no ICMP rules in use, that traffic can be ignored via a BPF.

```
TCP Port Flows
--------------
Port[22] 9.97% of Total, Src:  87.37% Dst:  12.63%
Port[25] 0.15% of Total, Src:   2.27% Dst:  97.73%
Port[80] 60.51% of Total, Src:  92.20% Dst:   7.80%
Port[143] 0.53% of Total, Src:  96.91% Dst:   3.09%
Port[443] 1.77% of Total, Src:  82.81% Dst:  17.19%
Port[993] 0.28% of Total, Src:  94.78% Dst:   5.22%
Ports[High<->High]: 26.26%


UDP Port Flows
--------------
Port[53] 6.22% of Total, Src:  61.89% Dst:  38.11%
Port[67] 28.60% of Total, Src:  48.01% Dst:  51.99%
Port[68] 28.60% of Total, Src:  51.99% Dst:  48.01%
Port[137] 10.62% of Total, Src:  49.89% Dst:  50.11%
Port[138] 6.42% of Total, Src:  50.00% Dst:  50.00%
Port[161] 0.26% of Total, Src:   0.00% Dst: 100.00%
Port[192] 0.50% of Total, Src:  15.31% Dst:  84.69%
```

```
Port[427] 0.36% of Total, Src:   0.00% Dst: 100.00%
Port[500] 8.23% of Total, Src:  50.00% Dst:  50.00%
Ports[High<->High]: 51.26%

ICMP Type Flows
---------------
Type[0] 7.72% of Total
Type[3] 43.82% of Total
Type[5] 0.17% of Total
Type[8] 48.15% of Total
Type[11] 0.12% of Total
```

Checklist/Quick Reference

Below is a checklist for using the perfmon statistics and flow data.

Perfmon Statistic	Component	Recommended Changes/Remedies
Pattern Match %	Detection	▪ Eliminate traffic from inspection via `ignore_ports`. ▪ Proper configuration of SSL, SSH, FTP/Telnet to ignore traffic. ▪ Use HTTP Inspect `client_flow_depth` and `server_flow_depth` options and DCE/RPC `max_frag_len` option. ▪ Use Stream5 TCP `dont_store_large_packets`. ▪ Use a BPF to pre-filter traffic from Snort.
Stream Session Cache Timeouts, Pruned TCP Sessions/Sec	Stream5	▪ Increase values for Stream5 `memcap` and `max_tcp`. ▪ Check SYNs vs SYN/ACKs to identify asymmertric routing environments.
Stream Session Cache Faults, TimedOut TCP Sessions/Sec	Stream5	▪ Increase values for Stream5 TCP `timeout`.
Frag Auto Deletes/Sec, Frag-Faults	Frag3	▪ Increase values for Frag3 `memcap`, `prealloc_memcap`, `prealloc_frags`, `max_frags`.
Frag Timeouts	Frag3	▪ Increase values for Frag3 `timeout`.
CPU Usage, Drop Rate	All	▪ Tune Snort's preprocessors and rules. ▪ Use preallocated memory for Frag3.
Flow Data	TCP/UDP Ports	▪ Eliminate traffic from inspection via `ignore_ports`. ▪ Proper configuration of SSL, SSH, FTP/Telnet to ignore

		traffic. ▪ Use a BPF to pre-filter traffic from Snort.
Flow Data	Large TCP Packet Sizes	▪ Use Stream5 TCP `dont_store_large_packets`.

Preprocessor & Rule Profiling

Profiling uses a close approximation of Snort's processing time in microseconds to measure how long various phases of detection take. This is affected by system load – if Snort is sharing CPU, profiling will not fully reflect the actual time taken.

Preprocessor Profiling

Despite the name, this is more than profiling preprocessors, as the profiling statistics breakdown each of the many phases of detection into smaller elements, including packet decoding, preprocessing and normalization (where each preprocessor is decomposed to various degrees), detection (fast pattern matching and individual rule options), and logging.

To improve overall performance, the desire is to reduce the average time spent on each packet, represented by the "total" line at the end. The profiling statistics show where Snort is spending significant amounts of time relative to each other. For example, if a large number of rules are being evaluated that all use PCRE, the PCRE portion will account for a much larger percentage of rule evaluation when compared to lesser used rule options.

Some statistics, for example, evaluation of Stream reassembled packets are significantly higher in terms of average microseconds (and percent of caller). Anywhere a line shows > 100% of caller, that is intentional and results from the time on that task not being included in the time for the caller. Unless Snort is terminated abnormally, the checks and exits columns should match.

A sample of preprocessor profiling statistics is shown below.

```
Preprocessor Profile Statistics (worst 100)
====================================================
 Num            Preprocessor Layer     Checks      Exits          Microsecs  Avg/Check Pct of Caller Pct of Total
 ===            ============ =====     ======      =====          =========  ========= ============= ============
  1                  detect     0   17414292   17414292          271204527      15.57         57.30        57.30
   1                   mpse     1   13621607   13621607          197538443      14.50         72.84        41.74
   2              rule eval     1   14033838   14033838           48991627       3.49         18.06        10.35
    1         rule tree eval     2   15167690   15167690           46237206       3.05         94.38         9.77
     1  preproc_rule_options     3   32484574   32484574            5725411       0.18         12.38         1.21
     2                  pcre     3     423215     423215            1992741       4.71          4.31         0.42
     3               content     3    1126848    1126848             705497       0.63          1.53         0.15
     4             byte_test     3    2335674    2335674             695179       0.30          1.50         0.15
     5                  flow     3    1892908    1892908             479103       0.25          1.04         0.10
     6              flowbits     3    1757876    1757876             351948       0.20          0.76         0.07
     7            uricontent     3     149306     149306             110861       0.74          0.24         0.02
     8               icmp_id     3     254979     254979              36397       0.14          0.08         0.01
     9                 itype     3      73993      73993              14158       0.19          0.03         0.00
    10              isdataat     3       1935       1935                249       0.13          0.00         0.00
    2               rtn eval     2     418409     418409             526055       1.26          1.07         0.11
  2                   decode     0   17487378   17487378           68055056       3.89         14.38        14.38
  3                       s5     0   17114282   17114282           40696764       2.38          8.60         8.60
   1                   s5tcp     1    4494249    4494249           16347963       3.64         40.17         3.45
    1             s5TcpState     2    4423467    4423467            8062964       1.82         49.32         1.70
     1             s5TcpData     3    2262122    2262122             909430       0.40         11.28         0.19
      1       s5TcpPktInsert     4      21690      21690             164331       7.58         18.07         0.03
     2             s5TcpFlush     3      16539      16539             447449      27.05          5.55         0.09
      1  s5TcpProcessRebuilt     4      16538      16538            2059846     124.55        460.35         0.44
```

2	s5TcpBuildPacket	4	16538	16538	23835	1.44	5.33	0.01
2	s5TcpNewSess	2	115269	115269	560566	4.86	3.43	0.12
4	DceRpcMain	0	12975854	12975854	13132216	1.01	2.77	2.77
1	DceRpcSession	1	12975854	12975854	9639261	0.74	73.40	2.04
5	frag3	0	188875	188875	8496960	44.99	1.80	1.80
1	frag3rebuild	1	93308	93308	255599	2.74	3.01	0.05
2	frag3insert	1	93369	93369	214576	2.30	2.53	0.05
6	httpinspect	0	2232525	2232525	5730661	2.57	1.21	1.21
7	perfmon	0	17488739	17488739	5170586	0.30	1.09	1.09
8	eventq	0	34867946	34867946	3789817	0.11	0.80	0.80
9	backorifice	0	12609034	12609034	2537827	0.20	0.54	0.54
10	ssl	0	4067788	4067788	1416584	0.35	0.30	0.30
11	smtp	0	577649	577649	722950	1.25	0.15	0.15
12	dns	0	2052546	2052546	244260	0.12	0.05	0.05
total	total	0	17379051	17379050	473276804	27.23	0.00	0.00

In the Preprocessor Profile Statistics, five columns are significant, the Checks, Microsecs, Average per check, Percent of caller, and Percent of total. The Preprocessor, Layer, and Num columns show the name of the preprocessor or detection phase, the call hierarchy, and the relative position to others within the same layer.

- Reducing a high number of checks can be achieved by correctly configuring the preprocessors to use the correct ports[1]. Properly configuring Snort to ignore traffic that it cannot inspect (for example, encrypted connections) will reduce both the number of checks for detection, but also for the related preprocessors.
- Reducing the average per check can be tougher, as that depends both on the size and content of the packet payload. Correctly managing the ports used for TCP reassembly will help Snort not needlessly evaluate large blocks of data for which there are no rules or preprocessors. The default list of TCP ports are related to a default set of protocols for which there are rules and/or preprocessors. In turning off a collection of rules or a preprocessor, the corresponding port should be removed from the TCP reassembly ports. For example if your system is not monitoring a network where there are FTP servers and you have turned off both ftp/telnet preprocessor and the FTP rules, port 21 should also be removed from the list of TCP reassembly ports.
- Eliminating unnecessary rules will reduce time for the mpse (multi pattern search engine) and rule evaluation portion of detect. Identifying unnecessary or overly expensive rules will be covered in the next section.

  If memory allows, use a faster pattern matching algorithm. Snort defaults to AC-BNFA, which uses less memory, but is 10% slower than AC – use `config detection: search-method ac` to switch pattern matchers.

[1] This also includes correctly identifying services if using the target-based feature.

Rule Profiling

Rule profiling provides a list of rules, by total expense, per evaluation expense, as well as indicating which rules resulted in alerts.

A sample of rule profiling statistics is shown below.

```
Rule Profile Statistics (all rules)
==========================================================
```

Num	SID	GID	Checks	Matches	Alerts	Microsecs	Avg/Check	Avg/Match	Avg/Nonmatch	Disabled
1	13827	3	3877459	0	0	4017024	1.0	0.0	1.0	0
2	13887	3	1059847	0	0	3915850	3.7	0.0	3.7	0
3	14783	3	5413027	0	0	3418035	0.6	0.0	0.6	0
4	15486	1	422103	0	0	3335514	7.9	0.0	7.9	0
5	6420	1	5413027	0	0	3124014	0.6	0.0	0.6	0
6	6456	1	5413027	0	0	3124014	0.6	0.0	0.6	0
7	4246	1	5413027	0	0	3124014	0.6	0.0	0.6	0
8	6444	1	5413027	0	0	3124014	0.6	0.0	0.6	0
9	6432	1	5413027	0	0	3124014	0.6	0.0	0.6	0
10	3171	1	5413027	0	0	2208664	0.4	0.0	0.4	0
11	4755	1	5413027	0	0	1713291	0.3	0.0	0.3	0
12	13825	3	3877459	0	0	1707927	0.4	0.0	0.4	0
13	7210	1	5413027	0	0	1498775	0.3	0.0	0.3	0
14	13948	1	1059847	582	0	1481278	1.4	5.0	1.4	0
15	15513	1	5413027	0	0	1471194	0.3	0.0	0.3	0
16	2511	1	5413027	0	0	1468578	0.3	0.0	0.3	0
17	2657	1	187692	0	0	1102051	5.9	0.0	5.2	0
18	2659	1	187692	0	0	1102051	5.9	0.0	5.2	0
19	2658	1	187692	0	0	1039742	5.5	0.0	4.9	0
20	2656	1	187692	0	0	1039721	5.5	0.0	4.9	0
21	15327	3	1059847	0	0	961025	0.9	0.0	0.9	0
22	2256	1	821156	0	0	805927	1.0	0.0	1.0	0
23	3059	1	187692	30	0	759668	4.0	4240.2	3.4	0
24	15462	3	11451	0	0	741080	64.7	0.0	64.7	0
25	2661	1	127225	38464	0	605107	4.8	9.5	2.7	0
26	13924	1	29533	0	0	205625	7.0	0.0	7.0	0
27	14782	3	73708	0	0	120335	1.6	0.0	1.6	0
28	13980	3	18401	0	0	109200	5.9	0.0	5.9	0
29	15262	1	19742	0	0	76854	3.9	0.0	3.9	0
30	14657	3	17408	0	0	57491	3.3	0.0	3.3	0
31	14656	3	17408	0	0	57334	3.3	0.0	3.3	0
32	14019	1	12804	0	0	44618	3.5	0.0	3.5	0
33	14020	1	12804	0	0	44603	3.5	0.0	3.5	0
34	15481	1	10609	0	0	40880	3.9	0.0	3.9	0
35	463	1	73993	0	0	39854	0.5	0.0	0.5	0
36	13465	1	10745	0	0	34344	3.2	0.0	3.2	0
37	13584	1	10745	0	0	32027	3.0	0.0	3.0	0
38	15294	1	10745	0	0	31114	2.9	0.0	2.9	0
39	224	1	73993	0	0	25916	0.4	0.0	0.4	0
40	228	1	73993	0	0	23204	0.3	0.0	0.3	0
41	251	1	73993	0	0	21328	0.3	0.0	0.3	0
42	5901	1	20358	0	0	18947	0.9	0.0	0.9	0
43	221	1	33000	0	0	18701	0.6	0.0	0.6	0
44	5800	1	20358	0	0	17945	0.9	0.0	0.9	0
45	13516	1	5820	0	0	13804	2.4	0.0	2.4	0
46	6409	1	10320	0	0	13570	1.3	0.0	1.3	0
47	6411	1	10320	0	0	11120	1.1	0.0	1.1	0
48	15386	3	2725	0	0	10486	3.8	0.0	3.8	0
49	6410	1	10320	0	0	10049	1.0	0.0	1.0	0
50	2278	1	440	0	0	9367	21.3	0.0	21.3	0
51	3084	1	1622	0	0	6714	4.1	0.0	4.1	0
52	7041	1	8787	0	0	6407	0.7	0.0	0.7	0
53	15683	3	196	0	0	6184	31.6	0.0	31.6	0
54	15149	3	3430	0	0	4591	1.3	0.0	1.3	0
55	15574	1	300	0	0	3959	13.2	0.0	13.2	0
56	15471	1	310	0	0	2822	9.1	0.0	9.1	0
57	1618	1	310	0	0	2574	8.3	0.0	8.3	0
58	3466	1	264	0	0	2499	9.5	0.0	9.5	0
59	2183	1	872	0	0	2107	2.4	0.0	2.4	0
60	12489	1	1665	0	0	2102	1.3	0.0	1.3	0
61	9027	1	1665	0	0	2102	1.3	0.0	1.3	0
62	9769	1	1665	0	0	1983	1.2	0.0	1.2	0
63	3461	1	136	0	0	1939	14.3	0.0	14.3	0
64	2597	1	132	0	0	1911	14.5	0.0	14.5	0

As with preprocessor profiling, many of the columns[2] are significant, the number of checks, microseconds, average per check (overall, match and non match), and the number of matches. Expensive rules are pretty easily identified with this data, usually by looking at the top 10 or 20 rules with high values in those columns. In particular, rules with an order of magnitude difference compared to others should jump out. The above table is sorted by microseconds, and the first 20 rules are an order of magnitude higher than the next group.

- A high value for microseconds for a given rule indicates that rule is costly. Comparing the total microseconds for a rule against the total microseconds for all of Snort provides an estimate of how much Snort's processing time per packet is spent on that rule. Because of the way rules are evaluated since Snort 2.8.2 – rule options from similar rules are evaluated once – the total number of microseconds for all rules will likely be more than the total processing time for all of Snort. A rule that has a ratio of greater than 5% of the total time should be evaluated to see if it can be altered to perform better and/or turned off for the given configuration. That is especially true if there are no matches or alerts – matches sometimes don't result in alerts because of a `flowbit:noalert` option.
- Looking at the total microseconds per rule relative to the number of checks and average per check will help identify how the rule may be altered. A large number of checks means that the content, if present, used for the fast pattern matcher is not significantly unique relative to the data being seen on the network. If there is no content, depending on the ports/services for the rule, it may be evaluated on every packet.

  A high average per check indicates a rule that has rule options that are expensive to evaluate, most often because of PCRE rule options with complex expressions. Breaking the rule up into multiple rules by reducing the complexity or eliminating the PCRE will reduce the evaluation time. For example, if a PCRE contains 3 "or'd patterns", use regular content options.
- The impact of PCRE can be limited by the `config` options `pcre_match_limit` and `pcre_match_limit_recursion`.
- The number of checks not only identifies rules that are being evaluated frequently, but when used in comparison to other rules, can be used to update the rules themselves. In the above example data is that a number of rules all have the same number of checks, implying that they use a common content option. If there are other, possibly more unique, contents with those rules, the `fastpattern` option could be added to the most unique content for each of those rules which will reduce the number of evaluations.

  A rule that is being evaluated frequently but never matching will result in a high average per non-match. These rules should be investigated for both accuracy and whether or not they need to be used at all.

[2] The disabled column is only present if the PPM feature is enabled.

Checklist/Quick Reference

Below is a checklist for using the preprocessor profiling statistics.

Preprocessor Profiling Statistic	**Issue**	**Recommended Changes/Remedies**
Checks	High relative to others	▪ Eliminate traffic from inspection via `ignore_ports`. ▪ Proper configuration of preprocessor ports and use target-based feature. ▪ Proper configuration SSL, SSH, FTP/Telnet to ignore traffic. ▪ Use a BPF to pre-filter traffic from Snort.
Microsecs, Avg per Check	High relative to others	▪ Proper configuration of preprocessor ports and use target-based feature. ▪ Correct configuration of TCP reassembly ports.
Detect, mpse	High (>50% total)	▪ Eliminate unnecessary rules, see below. ▪ Use faster pattern matcher algorithm (AC).
PCRE, Avg per Check	High, Large % of Total	▪ Use `config pcre_match_limit` and `config pcre_match_limit_recursion`

Below is a checklist for using the rule profiling statistics.

Rule Profiling Statistic	**Issue**	**Recommended Changes/Remedies**
Microsecs	> 5% Preprocessor Profiling "total" time	▪ Turn off rule if possible. ▪ Rewrite rule for better performance: o Use a content option (or identify most unique content with `fastpattern` option). o Correct source & destination ports and/or service metadata.
Matches and/or Alerts	Low Values	▪ Possible no alerts with high matches because of `flowbits:noalert`. Is flowbit necessary? ▪ Turn off rule if possible.
Avg per Check	High Value	▪ Identify expensive PCRE, possibly split rule. ▪ Ensure there is content. ▪ In conjunction with preprocessor profiling, identify

		expensive rule options. Update rule to be more efficient.
Checks	Same as other rules	▪ See if content options in these rules are the same. Use more unique content in different rules to differentiate them.

Change Log

Date	Author(s)	Summary of Changes
Aug 12, 2009	Sturges	Initial draft
Aug 17, 2009	Sturges	Added checklists, correct minor spelling errors.
Aug 19, 2009	Sturges	Updated TCP Timeout info, added section on perfmon flow stats
Oct 28, 2009	Sturges	Updated formatting