White Paper

Security

Performance Metrics for Java: Fortify Application Defender

For every IT person, understanding the performance impact created by a new solution brought into the production environment is invaluable to keep the servers up and running. In order to measure the performance overhead caused by Micro Focus® Fortify Application Defender, an application self-protection software, we tested the solution on a well-used open source enterprise automation software. Statistics show the execution time overhead range from 3 percent to 14 percent while the memory overhead is consistent around 4 percent.

MICRO FOCUS

Table of Contents

page

Magnolia CMS can run on a variety of application servers including IBM WebSphere, Oracle WebLogic, JBoss Application Server, and Apache Tomcat. It supports Derby DB, MySQL, Oracle, and PostgreSQL database.

Evaluation

The evaluation of the Application Defender was conducted on a well-used application called Magnolia CMS.[1] The application was brought up in an environment that was dedicated to performance testing and Apache JMeter was used to gather performance figures. In this section, we describe, in detail, the different aspects of the evaluation.

Real-World, Open Source Application

Magnolia[1,2] is one of the most popular open source content management systems (CMS) with customers like Allianz Insurance, United States Navy, ING Bank, Sony, Nissan, Virgin America Airlines, etc. 1 It is available under an open source license, the GPL version 3. Magnolia Community Edition includes an AJAX-powered intuitive web-browser interface, a clear Java programming API, and a useful custom tag library for easy templating in JSP and servlets. It offers Spring framework and Struts integration, RSS feed support, standard templates, data caching and backup, and more. As of February 2012, the Magnolia website lists over 70 free modules.

Magnolia CMS can run on a variety of application servers including IBM WebSphere, Oracle WebLogic, JBoss Application Server, and Apache Tomcat. It supports Derby DB, MySQL, Oracle, and PostgreSQL database.

Evaluation Goals

The goal of the test is to determine performance overhead Application Defender causes on the test application. We focused on measuring performance impact on executing time and memory consumption.

Application Defender supports two different rulepacks, namely Application Protection Rulepack and Application Logging Rulepack. Application Protection Rulepack is designed for detecting malicious attacks like SQL Injection, Cross-site Scripting, Command Injection, etc. while Application Logging Rulepack is designed for auditing security related events like user logon, user logoff, read/write a file, etc. In order to measure the performance impact under different configurations, we measured six different scenarios:

UNDER NORMAL TRAFFIC

1. Application Protection Rulepack
2. Application Logging Rulepack
3. Protection and Logging Rulepack

UNDER 6% ATTACK TRAFFIC

4. Application Protection Rulepack
5. Application Logging Rulepack
6. Protection and Logging Rulepack

Lab Environment

The test was driven from Apache JMeter which is a virtual machine running on a dedicated powerful blade server. The Magnolia CMS application is also a virtual machine running on another dedicated blade server. The third machine runs Application Defender management console and is responsible for managing

1 **www.magnolia-cms.com**/ *Magnolia CMS Community Edition: 5.2.5.*
2 **https://en.wikipedia.org/wiki/Magnolia**_*(CMS).*

Defender configuration and receiving events. The Magnolia CMS runs on Apache Tomcat 7 with embedded Derby database. See figure 1. Lab setup for the exact specifications.

All machines, operating systems, Magnolia CMS, and Application Defender were run with the default configurations. The only modifications were (1) enabling JMX on the Apache Tomcat to read the memory consumption remotely, and (2) changing the Apache Tomcat maximum heap memory to 3 GB.

All machines, operating systems, Magnolia CMS, and Application Defender were run with the default configurations. The only modifications were (1) enabling JMX on the Apache Tomcat to read the memory consumption remotely, and (2) changing the Apache Tomcat maximum heap memory to 3 GB.

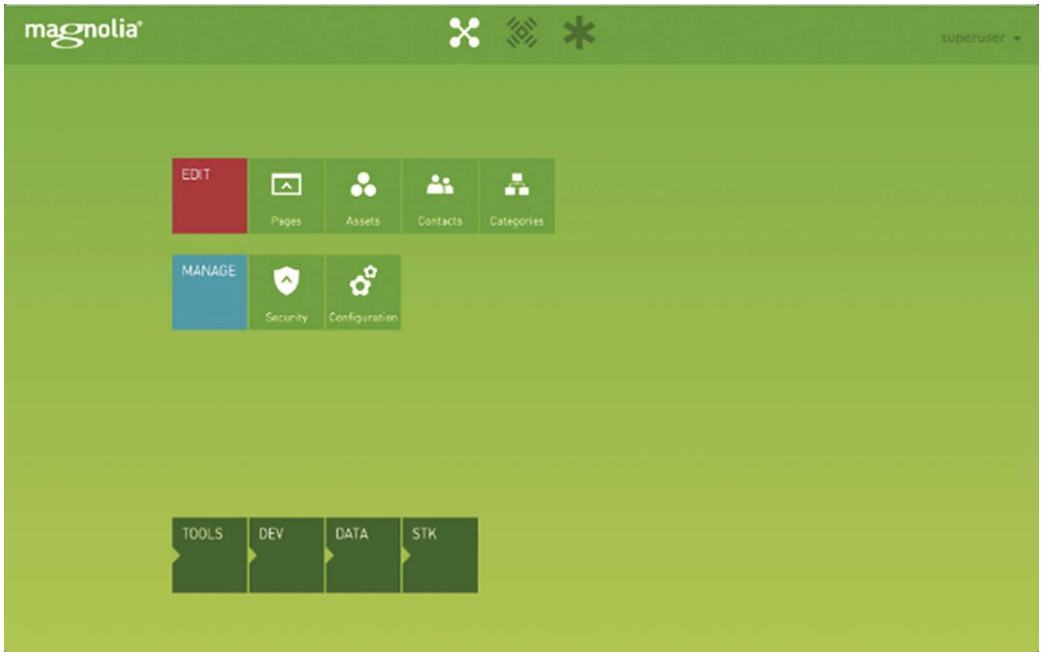

Figure 1. Lab Setup

All tests were conducted with Fortify Runtime Agent 16.5.0005 with 2016.2.2.6 Rulepack

Testing Methodology

We used Apache JMeter to measure the overhead caused by the Application Defender. We created a JMeter script with 28 GET and 35 POST requests, totaling 63 requests, all are dynamic/AJAX pages and does not include any images, static html, etc. We call this one loop. We executed this with 30 concurrent threads to simulate 30 concurrent users, and each thread executes 60 loops. Therefore, there are a total of (28+35) x 30 x 60 = 113,400 requests.[3] There are 2 extra loops (not counted as part of the 113,400 requests) for warming up the system to serve for tasks like class initialization, JSP compilation, instrumentation, etc. The first two warm up loops are noticeably slower than all the other loops. We first run one complete set of tests against the raw application, and we get the total execution time (t1) by measuring the time span between the first and the last JMeter request timestamps. Then, we run the same set of tests against the application with Application Defender, and we get the total execution time (t2). By comparing t1 and t2, we will then have the percentage of increase in execution time. Finally, this process is repeated 8 times, before we calculate the average with the best and worst numbers removed.

3 *Due to redirects and retries, the exact number of HTTP requests sent was more than 113,400.*

In order to test the performance of Application Defender under attack traffic, we also ran the same set of tests, with 4 extra attacks in each loop. Therefore, to measure the system performance under attack we generate 28 GET + 35 POST + 4 Attacks = 67 requests in each loop, which simulate 6 percent of attack traffic.

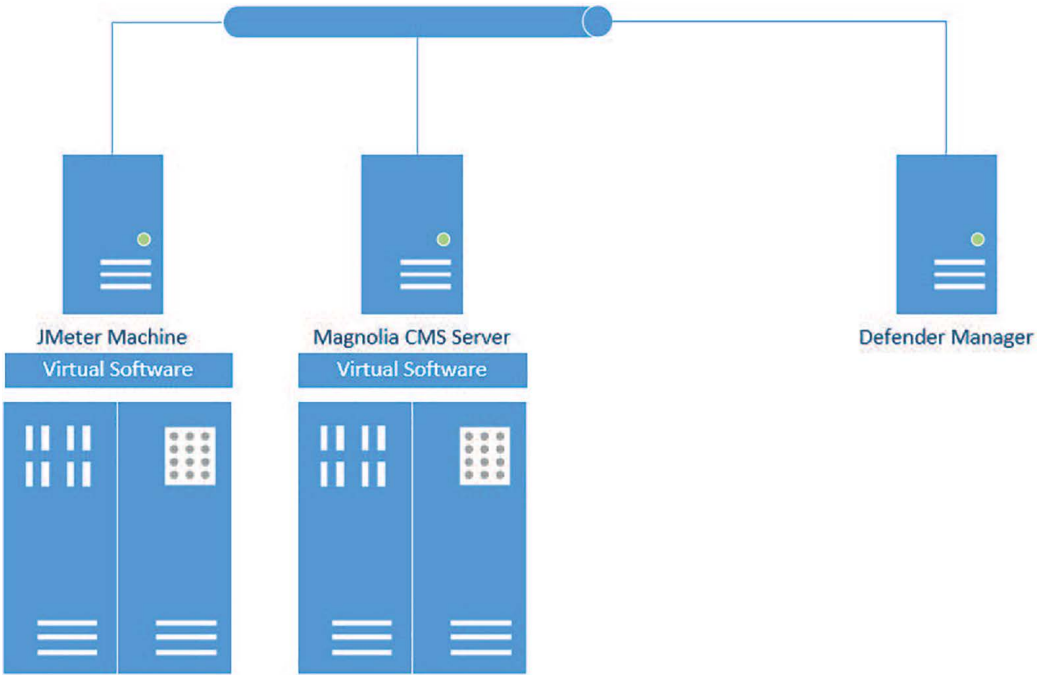

Figure 2. Number of loops executed

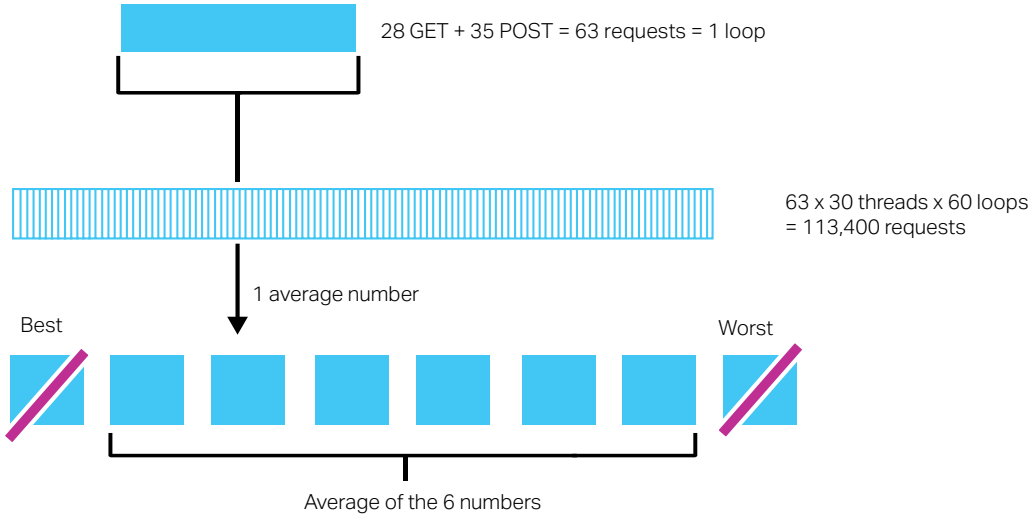

Figure 3. Number of loops executed

In order to test the performance of Application Defender under attack traffic, we also ran the same set of tests, with 4 extra attacks in each loop. Therefore, to measure the system performance under attack we

generate 28 GET + 35 POST + 4 Attacks = 67 requests in each loop, which simulate 6 percent of attack traffic. This test is not aimed at testing the performance of the protection rules because those rules will be executed even on normal traffic. This test measures the performance degradation caused by events generation and delivery.

We captured the memory consumption by enabling the Apache Tomcat JMX. Data was remotely captured and read every 5 seconds after the Apache Tomcat was started. At the end of the test, we compute the average memory consumption by taking the average of the middle three fifth of the data set (i.e. with first 1/5 and last 1/5 data ignored), and then compare the data with and without Application Defender installed. We compared the total heap and non-heap committed memory because this is almost identical to the total allocated memory as seen by the operating system.

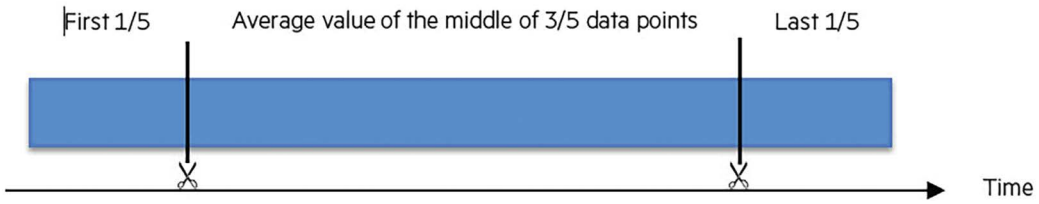

Figure 4. Compute of memory consumption

Results

Increase in Execution Time

Protection Rulepack Only Under Normal Traffic	Without App. Defender (Unit: ms)	With App. Defender (Unit: ms)	Increase
30Threads60Loops01	318,681	341,739	~~7.23%~~ (max)
30Threads60Loops02	331,306	344,077	3.91%
30Threads60Loops03	331,761	342,422	3.85%
30Threads60Loops04	330,138	340,988	3.47%
30Threads60Loops05	330,389	343,310	3.28%
30Threads60Loops06	330,047	340,440	3.27%
30Threads60Loops07	331,046	341,890	3.21%
30Threads60Loops08	332,669	342,461	3.14%
30Threads60Loops09	329,869	341,337	2.94%
30Threads60Loops10	331,663	341,335	~~2.91%~~ (min)
		Average (of 8)	**3.38%**

Protection Rulepack Only Under 6% Attack Traffic	Without App. Defender (Unit: ms)	With App. Defender (Unit: ms)	Increase
30Threads60Loops01	375,537	389,596	~~3.74%~~ (min)
30Threads60Loops02	376,419	391,544	3.77%
30Threads60Loops03	372,492	391,478	4.01%
30Threads60Loops04	373,869	387,984	4.31%
30Threads60Loops05	373,127	395,347	4.58%
30Threads60Loops06	371,769	391,027	4.69%
30Threads60Loops07	373,475	392,017	4.96%
30Threads60Loops08	373,036	390,135	5.09%
30Threads60Loops09	374,454	390,623	5.18%
30Threads60Loops10	372,782	390,274	~~5.95%~~ (max)
		Average (of 8)	**4.58%**

Logging Rulepack Only Under Normal Traffic	Without App. Defender (Unit: ms)	With App. Defender (Unit: ms)	Increase
30Threads60Loops01	331,921	356,275	~~5.90%~~ (min)
30Threads60Loops02	330,285	357,358	6.86%
30Threads60Loops03	329,675	356,037	7.12%
30Threads60Loops04	329,158	355,365	7.33%
30Threads60Loops05	333,443	357,212	7.69%
30Threads60Loops06	332,181	358,046	7.78%
30Threads60Loops07	335,310	355,096	7.96%
30Threads60Loops08	329,441	354,799	7.99%
30Threads60Loops09	330,538	357,678	8.19%
30Threads60Loops10	333,295	356,189	~~8.21%~~ (max)
		Average (of 8)	**7.62%**

Logging Rulepack Only Under 6% Attack Traffic	Without App. Defender (Unit: ms)	With App. Defender (Unit: ms)	Increase
30Threads60Loops01	376,520	404,877	~~7.53%~~ (min)
30Threads60Loops02	375,565	405,717	7.84%
30Threads60Loops03	374,431	405,164	7.97%
30Threads60Loops04	372,373	405,663	8.02%
30Threads60Loops05	374,811	409,898	8.14%
30Threads60Loops06	374,009	403,837	8.20%
30Threads60Loops07	374,256	404,735	8.71%
30Threads60Loops08	372,709	405,189	8.93%
30Threads60Loops09	375,021	404,449	9.14%
30Threads60Loops10	372,244	406,291	~~9.36%~~ (max)
		Average (of 8)	**8.38%**

Protection and Logging Under Normal Traffic	Without App. Defender (Unit: ms)	With App. Defender (Unit: ms)	Increase
30Threads60Loops01	332,756	368,746	~~12.04%~~ (max)
30Threads60Loops02	332,388	368,269	12.04%
30Threads60Loops03	330,280	370,069	11.94%
30Threads60Loops04	330,600	370,421	11.75%
30Threads60Loops05	330,736	368,595	11.44%
30Threads60Loops06	331,310	370,888	10.81%
30Threads60Loops07	334,009	368,234	10.79%
30Threads60Loops08	331,886	367,693	10.78%
30Threads60Loops09	332,507	371,607	10.37%
30Threads60Loops10	333,769	368,390	~~10.24%~~ (min)
		Average (of 8)	**11.24%**

Protection and Logging Under 6% Traffic	Without App. Defender (Unit: ms)	With App. Defender (Unit: ms)	Increase
30Threads60Loops01	375,004	425,184	~~15.77%~~ (max)
30Threads60Loops02	374,618	428,045	15.03%
30Threads60Loops03	ERROR		
30Threads60Loops04	371,553	430,182	14.65%
30Threads60Loops05	372,312	423,808	14.26%
30Threads60Loops06	377,730	427,211	14.16%
30Threads60Loops07	373,341	426,241	13.83%
30Threads60Loops08	371,827	427,730	13.80%
30Threads60Loops09	373,047	427,709	13.38%
30Threads60Loops10	373,326	424,850	~~13.09%~~ (min)
		Average (of 7)	**14.16%**

Increase in Memory

Protection Rulepack Only Under Normal Traffic	Without App. Defender (Unit: Mega Byte)	With App. Defender (Unit: Mega Byte)	Increase
30Threads60Loops01	Error: JMX connection failed	2142	N/A
30Threads60Loops02	2051	2145	~~5.21%~~ (max)
30Threads60Loops03	2036	2142	4.93%
30Threads60Loops04	2052	2133	4.67%
30Threads60Loops05	2061	2113	4.58%
30Threads60Loops06	2080	2129	3.95%
30Threads60Loops07	2036	2131	3.68%
30Threads60Loops08	2027	2127	2.52%
30Threads60Loops09	2067	2143	2.37%
30Threads60Loops10	2070	2119	~~2.36%~~ (min)
		Average (of 7)	**3.81%**

Protection Rulepack Only Under 6% Attack Traffic	Without App. Defender (Unit: Mega Byte)	With App. Defender (Unit: Mega Byte)	Increase
30Threads60Loops01	2145	2156	~~0.51%~~ (min)
30Threads60Loops02	2101	2147	0.99%
30Threads60Loops03	2101	2165	1.32%
30Threads60Loops04	2081	2170	2.19%
30Threads60Loops05	2064	2163	2.26%
30Threads60Loops06	2126	2147	2.51%
30Threads60Loops07	2126	2154	3.05%
30Threads60Loops08	2122	2170	3.69%
30Threads60Loops09	2112	2165	4.28%
30Threads60Loops10	2113	2191	~~4.80%~~ (max)
		Average (of 8)	**2.54%**

Logging Rulepack Only Under Normal Traffic	Without App. Defender (Unit: Mega Byte)	With App. Defender (Unit: Mega Byte)	Increase
30Threads60Loops01	2073	2107	~~0.43%~~ (min)
30Threads60Loops02	2074	2122	1.64%
30Threads60Loops03	2063	2120	1.68%
30Threads60Loops04	2087	2125	1.82%
30Threads60Loops05	2033	2128	2.31%
30Threads60Loops06	2095	2104	2.76%
30Threads60Loops07	2035	2121	2.94%
30Threads60Loops08	2083	2118	4.03%
30Threads60Loops09	2073	2134	4.23%
30Threads60Loops10	2035	2117	~~4.67%~~ (max)
		Average (of 8)	**2.68%**

Logging Rulepack Only Under 6% Attack Traffic	Without App. Defender (Unit: Mega Byte)	With App. Defender (Unit: Mega Byte)	Increase
30Threads60Loops01	2062	2139	~~5.69%~~ (max)
30Threads60Loops02	2057	2174	5.32%
30Threads60Loops03	2116	2137	3.94%
30Threads60Loops04	2065	2124	3.73%
30Threads60Loops05	2049	2158	3.47%
30Threads60Loops06	2138	2141	2.86%
30Threads60Loops07	2057	2138	2.59%
30Threads60Loops08	2081	2135	2.14%
30Threads60Loops09	2103	2176	0.99%
30Threads60Loops10	2104	2149	~~0.14%~~ (min)
		Average (of 8)	**3.13%**

Protection and Logging Under Normal Traffic	Without App. Defender (Unit: Mega Byte)	With App. Defender (Unit: Mega Byte)	Increase
30Threads60Loops01	2094	2148	~~5.44%~~ (max)
30Threads60Loops02	2065	2164	4.90%
30Threads60Loops03	2041	2152	4.79%
30Threads60Loops04	2070	2166	4.64%
30Threads60Loops05	2055	2123	3.80%
30Threads60Loops06	2041	2141	3.71%
30Threads60Loops07	2084	2132	3.31%
30Threads60Loops08	2049	2125	3.26%
30Threads60Loops09	2079	2158	2.58%
30Threads60Loops10	2087	2155	~~2.30%~~ (min)
		Average (of 8)	**3.87%**

Protection and Logging Under 6% Attack Traffic	Without App. Defender (Unit: Mega Byte)	With App. Defender (Unit: Mega Byte)	Increase
30Threads60Loops01	2120	2155	~~1.65%~~ (min)
30Threads60Loops02	2073	2168	1.69%
30Threads60Loops03	2064	2160	2.73%
30Threads60Loops04	2113	2177	3.02%
30Threads60Loops05	2118	2190	3.03%
30Threads60Loops06	2074	2190	3.40%
30Threads60Loops07	2121	2185	3.41%
30Threads60Loops08	2123	2181	4.58%
30Threads60Loops09	2113	2185	4.65%
30Threads60Loops10	2129	2165	~~5.59%~~ (max)
		Average (of 8)	**3.31%**

The reason for the memory overhead under 6 percent attack traffic (2.05 percent) is smaller than the figure without attack traffic (2.42 percent) is probably due to tolerance error. Measuring the exact memory consumption is very difficult because most objects are short-lived and the presence of the garbage collector makes the exact value a bit indeterministic.

Summary and Conclusion

Under Normal Traffic	Execution Time Overhead	Memory Overhead
Protection Only	3%	4%
Logging Only	8%	3%
Protection and Logging	11%	4%

Under 6% Attack Traffic	Execution Time Overhead	Memory Overhead
Protection Only	5%	3%
Logging Only	8%	3%
Protection and Logging	14%	3%

As expected, the execution time overhead for “Protection and Logging” roughly equals to “Protection” and added add together. The Logging rulepack generates around 524 events per loops, which is about

8.3 events per page, with the majority (~78%) of them being "Database Query" events. Obviously, disabling some of these logging rules will generate fewer events and improve the overall performance. As for Protection rulepack, it performs more tasks when the application is under attack and this was reflected in the extra 2 percent degradation when the application was under 6% attack traffic.

Both Protection and Logging rulepacks require to hold very little or no extra information when doing the runtime analysis. All the objects created during the analysis are very short lived and therefore, the impact on the overall memory overhead is expected to be very low. When further investigating the memory data, we found that there was constantly about a 20MB increase in PermGen memory consumption when Application Defender was installed. See Figure 3 for the PermGen memory of the fourth data set of Logging only rulepack under 6% attack traffic. Charts on other data set shown similar characteristics. Because this happened right after the server started up, we believe this 20MB memory overhead is due to extra classes or libraries loaded by the Defender agent and this figure will remain roughly the same value even among different applications and different Defender configurations. The rest of the 4% increase in memory, roughly 40MB in actual figure, is probably due to extra memory consumption due to detection analysis its corresponding short-lived objects.

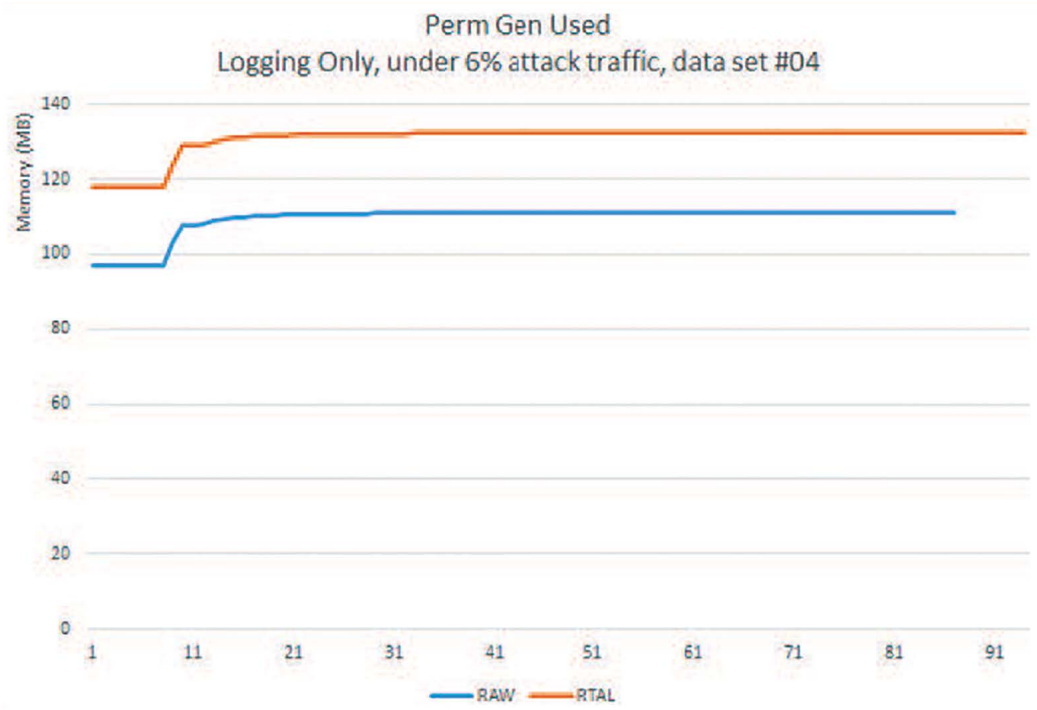

Figure 5. PermGen memory consumption

Learn more at
www.microfocus.com/rasp

Contact us at:
www.microfocus.com

Like what you read? Share it.