

Laboratory Report

An Appendix to “SELinux & grsecurity: A Side-by-Side Comparison of Mandatory Access Control & Access Control List Implementations”

1. Hardware Configuration

We configured our testbed on two identical PC platforms obtained from the Computer Science Department’s Systems Staff (sunny.cs.virginia.edu & cloudy.cs.virginia.edu). Each PC has the following attributes:

CPU- AMD K6 II 3D (i586) at 450MHz
Memory- 128MB PC100 DIMM by Simple Technologies
HD- 8Gb Fujitsu Model # MPE3084AE

These two PCs are connected to the UVA network out of the Reynolds Lab in Small Hall.

2. Operating System & Security Patches

Gentoo Linux Distribution Release 1.4 for x86 architectures

-SELinux on sunny.cs.virginia.edu
-grsecurity on cloudy.cs.virginia.edu

3. Performance Benchmarking

In order to provide a relative comparative basis for analyzing the performance of the security patches, we selected 2 benchmarking suites – UnixBench 4.1.0 and LMBench. These tools provide quantitative results for a number of microbenchmarks included in each suite. The selection of benchmarks was intended to replicate experiments conducted by Loscocco and Smalley on SELinux in [13], apply those experiments to grsecurity and compare performance results to a baseline unmodified kernel, and each OS patch relative to the other in a side-by-side comparison.

3a. Unixbench 4.1.0

Most Unixench microbenchmarks execute a number of iterations and report results for a particular benchmark as an average of those iterations. In order to provide a greater degree of normalization, we chose to run the Unixbench suite three times each in various user modes (i.e. – root user unenforced, root user enforced, regular user unenforced, etc) on both testbed systems. The results from the three runs have been compiled and we report the average of these runs. The Unixbench benchmark suite includes the following microbenchmarks that we selected to include in our report:

Whetstone- assignment, addition, subtraction & multiplication calculations that substitute datatypes for numbers (register, short, int, float, long, double and an empty loop)

Dhrystone 2- manipulation of arrays, character strings, indirect addressing, and most of the non-floating point instructions that might be found in an application program. It also includes conditional operations and other common program flow controls.

Execl- replacing a currently running process with a new process

File copy 4K/1K/256B buffer sizes- captures the number of characters that can be copied within 10 seconds based on varying buffer sizes

Pipe throughput- a single process opens a pipe (an inter-process communications channel) to itself and spins a megabyte around a short loop.

Context-switching- a test program spawns a child process with which it carries on a bi-directional pipe conversation.

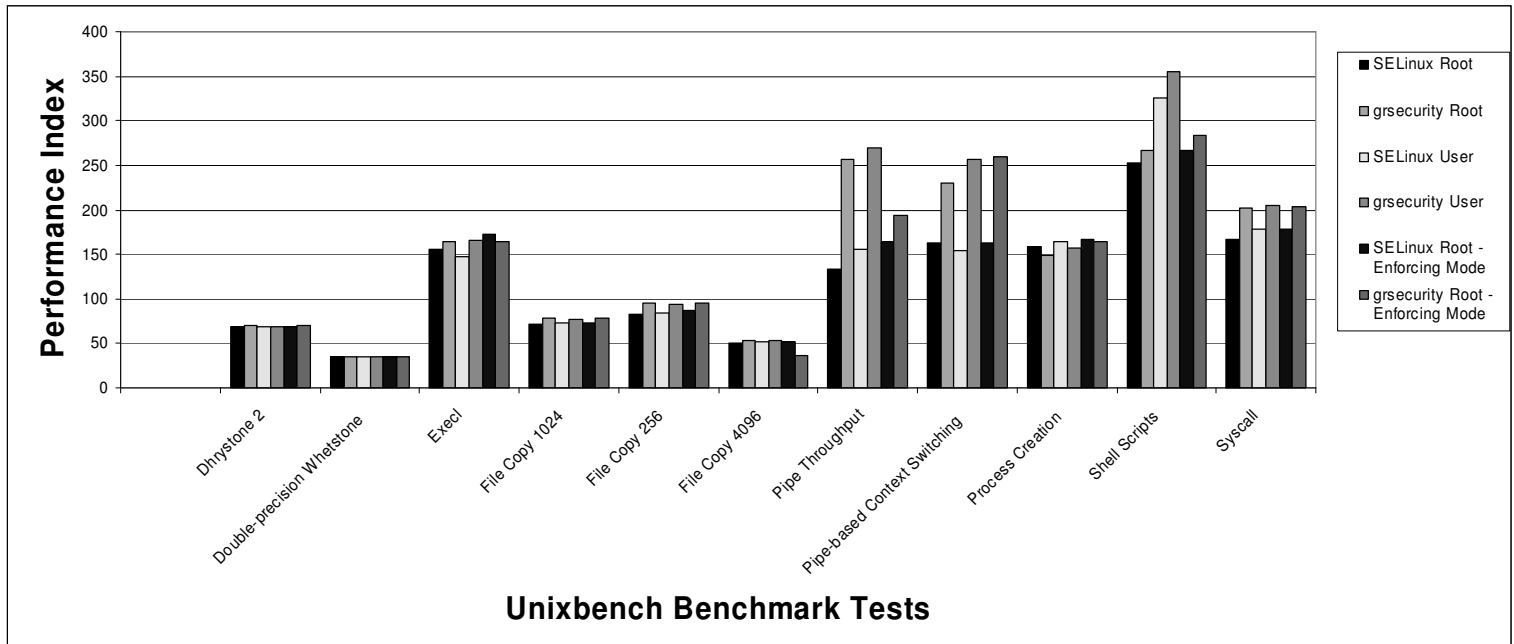
Process creation- a test that creates a child process which immediately dies after its own fork(). The process is repeated over and over.

Shell scripts- a shell script that is run by 1, 2, 4, and 8 concurrent processes. The script consists of an alphabetic sort one file to another; taking the octal dump of the result and doing a numeric sort to a third file; running grep on the result of the alphabetic sort file; “tee”ing the result to a file and to wc (wordcount); writing the final result to a file; and removing all of the resulting files.

Syscall- this test evaluates the time required to do iterations of dup(), close(), getpid(),getuid(), and umask() calls.

As described above, the averaged results yielded by UnixBench are included in Figure 1. While a detailed interpretation of these results requires understanding of what metric each benchmarks reports, higher index numbers indicate better performance.

	<u>Root User</u>		<u>User</u>		<u>Root-Enforcing Mode</u>	
	<u>SELinux</u>	<u>grsecurity</u>	<u>SELinux</u>	<u>grsecurity</u>	<u>SELinux</u>	<u>grsecurity</u>
Dhrystone 2	69.4	69.5	69.4	69.2	69.4	69.5
Whetstone	35.4	35.4	35.4	35.4	35.4	35.4
Execl	155.6	163.6	147.9	165.6	172.9	164.2
File Copy 1K	71.6	79.3	72.5	77.5	73.6	78.1
File Copy 256	83.3	95.8	84.8	94.7	87.3	95.7
File Copy 4K	50.8	53.1	51.3	53.2	51.8	36.7
Pipe Throughput	134.0	256.7	155.3	268.9	164.6	194.0
Context Switching	162.8	230.3	154.2	257.3	162.4	259.9
Process Creation	159.0	149.5	164.2	156.7	167.5	163.9
Shell Scripts (8)	252.4	266.7	325.8	355.0	267.2	283.9
Syscall	166.5	202.2	178.1	205.5	178.7	203.0



Observation #1- Unixbench uses a shell command, `time`, in order to aggregate timing performances for each of its benchmarks. Along with counting execution loops of each microbenchmark, the timings are used by Unixbench in order to derive performance index scores for each microbenchmark. After installing and building the Unixbench source code, I determined that the benchmark suite was not generating program timings the way that it should. After tracing the timing datatype through the source code, and thoroughly examining the user documentation, I discovered that there are at least 3 versions of the shell program `time`, all of which report a string in different formats. The version of `time` included in the Bourne again shell with our Gentoo Linux distribution was not compatible with the Unixbench source code, and this concern was not addressed in the documentation or in user groups. After installing a GNU version of `time`, Unixbench gathered program timings correctly.

Observation #2- Loscocco & Smalley used version 4.1.0 of Unixbench to analyze SELinux, and while I initially chose an earlier, more stable release of Unixbench (v4.0.1) to install and test with, the microbenchmarks have been altered slightly, so I updated our Unixbench install to 4.1.0 in order to provide a more accurate statistical comparison to earlier work and ensure results are uniform.

Observation #3- As the benchmarks were conducted before much of the policy enforcement work was done on our tested systems, I discovered that the default policies included in grsecurity allowed me to at least invoke enforcing mode either at the workstation or remotely, and still execute programs owned by the user. The same was not true of SELinux. Even as root, once I entered enforcing mode with default policies, the workstation would lock. With modified permissions, invoking the benchmarks remotely was difficult, as ssh was usually disabled.

Observation #4 – It is generally not wise to execute a `kill -9 -1` remotely when acting as root. Having never been a remote root user, I executed this shell command and disabled a number of processes required for remote access. I had to physically reboot the workstation in order to resume testing.

3b. LMBench 2.0.4

As opposed to Unixbench which reports indexed results of microbenchmarks performed during execution, Lmbench simply reports program timings in microseconds for each microbenchmark. As with Unixbench, we executed each benchmark suite three times each in various modes. This provides for greater normalization, and

helps stratify performance metrics between user & administrator modes. The microbenchmarks we selected for inclusion in our report are:

syscall measures how long it takes to write one byte to /dev/null

read measures the execution time of a fixed number of reads of four byte integers to memory

write does the same as read, but times the writing of the integers

fstat executes the system call *fstat* on a temporary file which has been open for read, then measures the time required to get file status.

stat does the same with a file in the file system that is not open.

open/close times the opening & closing of a temporary file for reading.

Pipe latency, AF_UNIX, UDP & TCP latency and TCP/IP all time inter-process communication between two processes

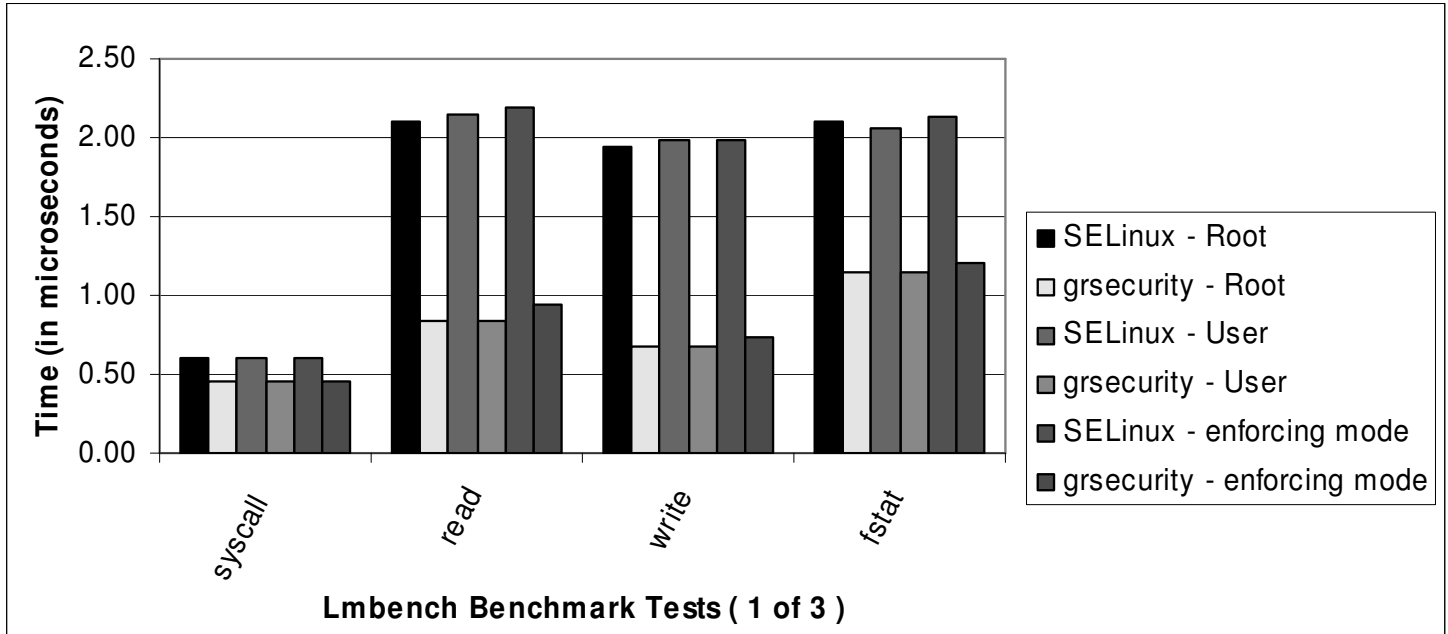
Fork, execve and /bin/sh time all time process creation, ranging from simple process creation “fork() and exit”, to the more expensive “fork() and execve”, to the most expensive “fork() and instantiate the shell”.

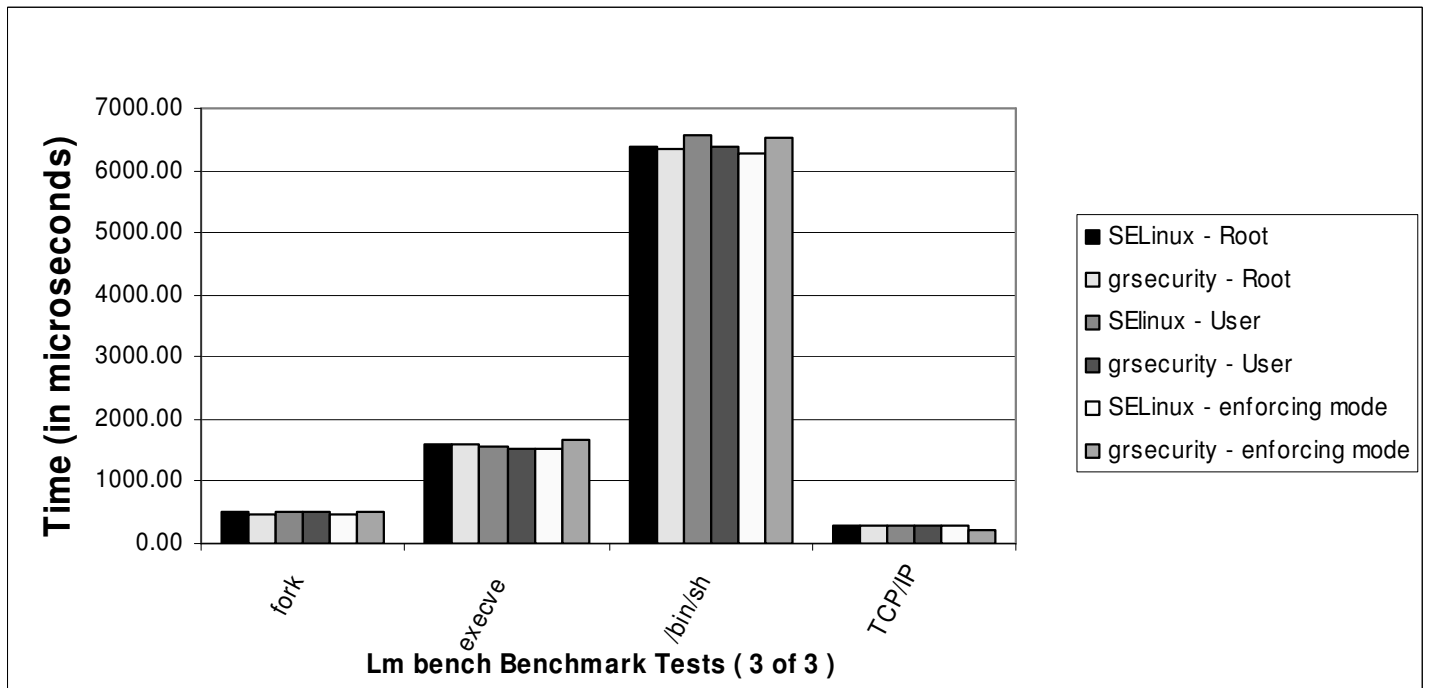
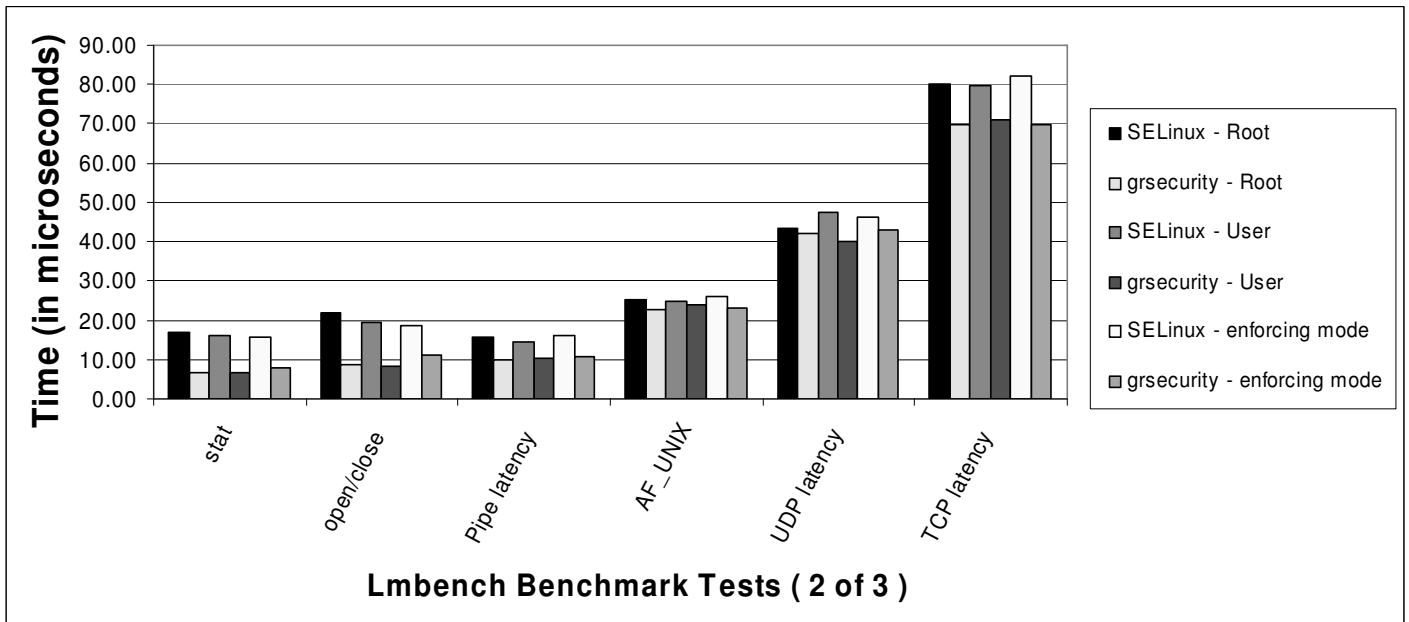
Program timings in Lmbench, measured in microseconds, indicate better performance with a lower time – exactly the opposite way of interpreting Unixbench results.

Observation #1 As noted above the results reported by Unixbench and Lmbench are opposite in terms of comparison. While we can extract the program timings for Unixbench, the extra layer of abstraction allows for comparison across a more uniform scale, as opposed to Lmbench, where results differ in several orders of magnitude. A similar approach to Unixbench would make Lmbench results easier to interpret and compare.

Observation #2- Lmbench generates a single results log for each iteration of the benchmark suite, whereas Unixbench divides results into three files – a log file, a report file and a times files. The report file is well formatted with easy to read tables and columns that report raw data and the index scores as well. This file is easy to read and makes interpreting results much easier. Lmbench’s approach is a little less straightforward. Since all results are piped to the same file, it is very crowded with data after the first 100 lines or so. From these first lines of the log, however, the user can extract the results of the pertinent microbenchmarks, although they are not tabulated or listed in columns. When extracting results from tens of files, this creates additional overhead on the user.

	<u>Root User</u>		<u>User</u>		<u>Enforcing Mode - Root</u>	
	<u>SELinux</u>	<u>grsecurity</u>	<u>SELinux</u>	<u>grsecurity</u>	<u>SELinux</u>	<u>grsecurity</u>
syscall	0.60	0.46	0.60	0.46	0.60	0.46
Read	2.10	0.84	2.14	0.84	2.20	0.95
Write	1.94	0.68	1.98	0.68	1.98	0.74
Stat	16.97	6.79	15.92	6.51	15.59	7.92
Fstat	2.10	1.15	2.05	1.15	2.13	1.20
Open/close	21.85	8.72	19.37	8.46	18.45	10.96
Fork	505.82	483.26	490.64	521.82	483.60	492.36
execve	1592.83	1579.92	1555.42	1520.58	1533.81	1661.00
/bin/sh	6393.33	6340.33	6576.00	6373.00	6278.33	6529.67
Pipe latency	15.82	9.93	14.41	10.32	15.91	10.69
AF_UNIX	25.39	22.55	24.58	23.89	25.82	23.21
UDP latency	43.53	42.23	47.51	39.86	46.44	43.04
TCP latency	79.93	69.95	79.74	71.07	82.10	69.83
TCP/IP	286.38	283.20	287.10	281.19	281.31	218.49





4 Conclusions & Supporting Data

Conclusions based on the benchmark results are reported in the accompanying paper, as well as analysis of the causes of any overhead introduced by the security patches. Supporting data generated by our experiments are included as an addendum to this report. As of the submission of our work, the testbed remains configured and accessible.