Contribution ID: **257** Type: **not specified**

Using a cBPF Binary Tree in Libseccomp to Improve Performance

Tuesday, November 13, 2018 4:25 PM (15 minutes)

Several in-house Oracle customers have identified that their large seccomp filters are slowing down their applications. Their filters largely consist of simple allow/deny logic for many syscalls (306 in one case) and for the most part don't utilize argument filtering.

Currently libseccomp generates an if-equal statement for each syscall in the filter. Its pseudocode looks roughly like this:

```
if (sycall == read)
     return allow
 if (syscall == write)
     return allow
 ...
 \# 300 or so if statements later
 if (syscall == foo)
     return allow
 return deny
```

This is *very* slow for syscalls that happen to be at the end of the filter. Libseccomp currently allows prioritizing the syscalls to place the most frequent ones at the top of the filter, but this isn't always possible - especially in a cloud situation.

I currently have a working prototype and the numbers are very promising.
In this prototype, I timed calling getppid() in a loop using a filter similar to one of my customer's seccomp filters. I ran this loop one million times and recorded the min, max, and mean times (in TSC ticks) to call getppid().
(I didn't disable interrupts, so the max time was often large.) I chose to report the minimum time because I feel it best represents the actual time to traverse the syscall.

The code for the libseccomp RFE is available here:
https://github.com/seccomp/libseccomp/issues/116

Test Case minimum TSC ticks to make syscall

seccomp disabled 138
getppid() at the front of 306-syscall seccomp filter 256
getppid() in middle of 306-syscall seccomp filter 516
getppid() at the end of the 306-syscall filter 1942
getppid() in a binary tree 312

I agree to abide by the anti-harassment policy

Presenter: HROMATKA, Tom (Oracle)

Session Classification: Containers MC