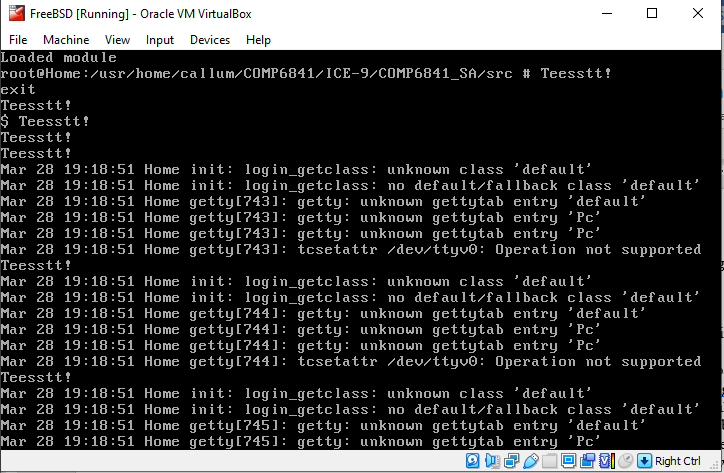
Writing Function Hooking:

Having huge issues finding read, mkdir, etc.

* So read, mkdir are now no longer the name of the system calls. Instead, this is calling the C standard library functions, which can only be run in the userspace.
* We need sys\_(name). See:
  + r225617 | kmacy | 2011-09-16 06:58:51 -0700 (Fri, 16 Sep 2011) | 12 lines  
      
    In order to maximize the re-usability of kernel code in user space this patch modifies makesyscalls.sh to prefix all of the non-compatibility calls (e.g. not linux\_, freebsd32\_) with sys\_ and updates the kernel entry points and all places in the code that use them. It also fixes an additional name space collision between the kernel function psignal and the libc function of the same name by renaming the kernel psignal kern\_psignal(). By introducing this change now we will ease future MFCs that change syscalls.
* And this fixed it!

So now, onto my function hooking:

* See this github link for the code:
* Right now, it simply overrides read() with a dummy print function. This is great because IT WORKS. It’s not so great in that I essentially nuked read(), so when I run the rootkit the entire OS gets bricked. Very nice.
* Interestingly, it seems to print my debug statement on a loop. There must be some function continuously reading from some source, so it could be interesting to look more into this later.
* 
* As we can see, the function hook works! (In a manner of speaking). My guess is getty or something is just trying to read constantly (we can see it enters some process and never leaves).
* I’m calling this a victory today. Next session, I’m going to clean up the actual content of the hook so it works as intended (hiding some pretend virus files from a user), and then on to hiding the actual rootkit from user detection.