Brow1325
Connor Brown

Bonus:

a. In Problem 4, lab4, an attacker modified the return address of ctxsw() to jump to its malware code which then behaved overtly or covertly depending on the attacker's objective. In the bonus problem, consider mounting a defense in the context of XINU running on galileo backends subject to the attacker of lab4. The solution cannot use a canary to detect corruption since return address overwrite was performed surgically without collateral corruption of surrounding memory. Instead, think about a ROP based defense where kernel code is modified to check if the return address of ctxsw() (and similarly for resched() and sleepms()) is valid, i.e., safe to jump to, and does so only if it is. Perhaps a modified return address may even be corrected to its original value so that jumping is feasible despite an attack. Describe a detailed solution in Lab5Answers.pdf in lab5/. There is no need to implement your solution.
    i. One possible solution could be to randomize the layout of the addresses. By that I mean we have it when XINU starts that it starts at a random address in memory that way they are always in different locations making it harder to locate the addresses. However this would not be perfect because it would still load in the same order just at a different address. So we can add further randomization by randomizing where the stacks are located and place them in separate locations in memory. By doing this we further isolate and protect each stack making it harder to modify the address values.