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.

Before ctxsw() is called for each process, there can be a check to ensure that the return address is safe to jump to. We can check to make sure that the return address is within the text segment of memory to ensure that the code we are jumping to is within xinu's kernel code. If the return address is not within these specified bounds, then do not proceed to return. One way that we can ensure that program continues to operate without returning to malicious code is to add a global variable that is constantly updated to contain the expected return address. If the expected return address does not match the actual return address, then just overwrite the return address to be the expected return address.