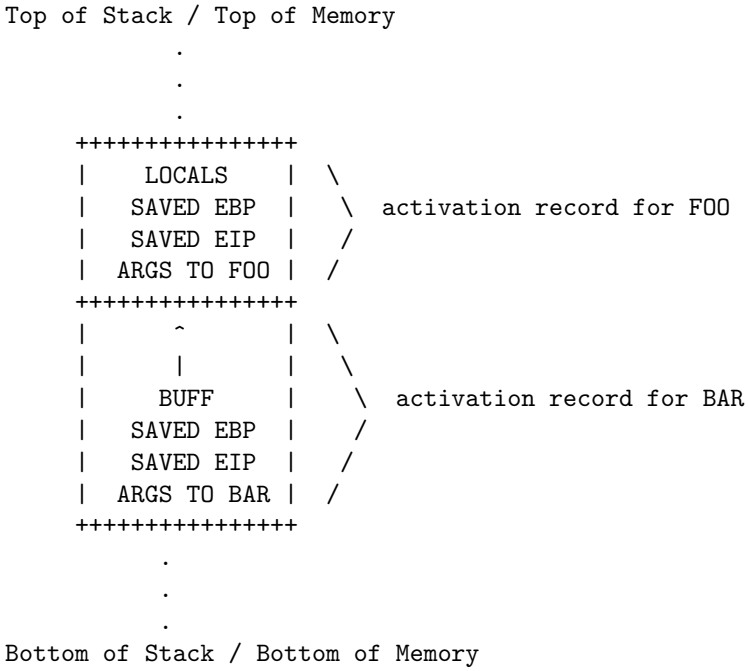
CS 155
Homework 1

Rafael Moreno Ferrer
SUID#: 05490330

Problem 1. *(Control Hijacking)*

a) In x86 the stack grows downwards. Explain how a stack-based overflow attack would work if the stack grew upwards instead.

For upward growing stacks overflowing an insecure buffer in the current activation record[1] will not generally result in a stack smashing attack since, at most, we will overwrite local variables but not any of the control data embedded in the stack (like saved eips, ebps or arguments). **However, stack smashing attacks can be done in this architecture by overflowing a buffer in a previous activation record.**



Suppose function BAR calls function FOO and function BAR has a local buffer BUFF. Inside function FOO there is code that copies data into BUFF unsafely. BUFF then can be overflown to overwrite the SAVED EIP for FOO and hijack control when FOO returns.

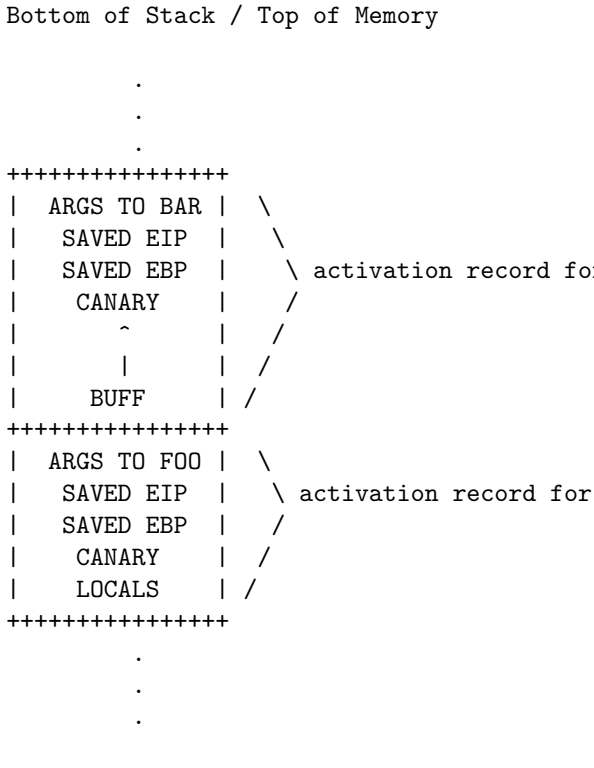
b) How would you implement StackGuard in an architecture where the stack grows upwards? What would be different from StackGuard on the x86?

From the picture above we see that in order to overflow BUFF and overwrite the SAVED EIP for FOO we need to stomp first over ARGS TO FOO and then its SAVED EIP. To implement StackGuard in the upward growing architecture we can make the compiler **place a canary in the stack right before the arguments for a function call to the stack. Prior to function return we will verify the integrity of the current activation record by checking the canary** and kill the process if we detect corruption. This differs from x86

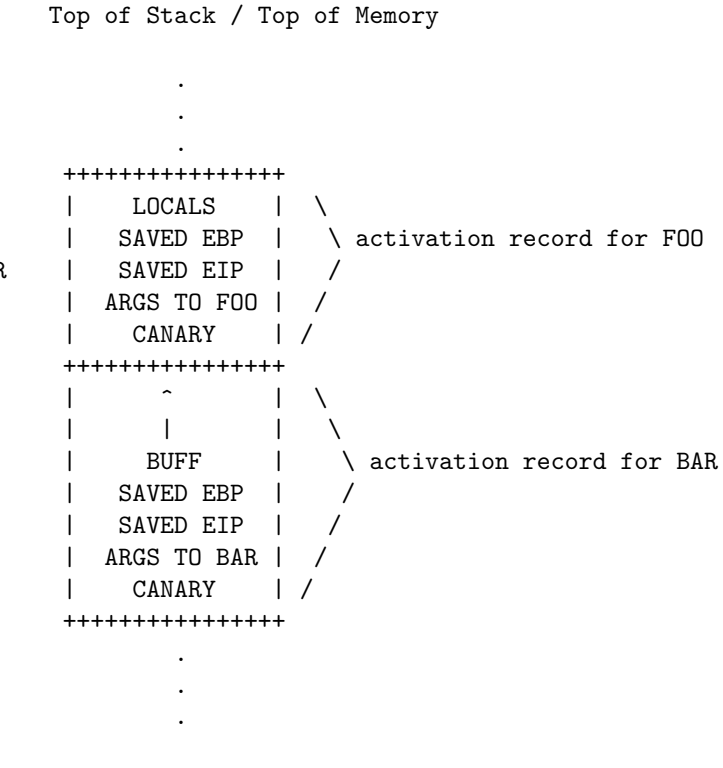
[1] the activation record of the function being cureently executed

in fact that canary in x86 has to defend against a buffer in an activation record overwriting the saved eip of that activation record as opposed to a buffer in previous activation record corrupting the saved eip of a subsequent activation record. Thus the placement of canary has to be different.

a) StackGuard layout for downward growing stack



b) StackGuard layout for upward growing stack



Problem 2.

Problem 3.

Problem 4.

Problem 5.

Problem 6.