Matthew Kachar

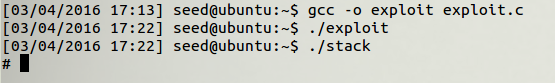
COSC4931

Homework 6 – Buffer Overflow Vulnerability

Objective: Understanding buffer overflow; when data is written outside the boundaries of the allocated memory space. This kind of vulnerability can result in loss of program control and program data.

Since Linux randomizes the heap and the stack location for the given kernel, it is easier to accomplish this lab by initializing the address to 0. Also it is necessary to disable the overwriting flag with the command –fno-stack-protector when compiling stack.c. Without these sort of protection overflow onto critical data on the stack becomes a very real issue.

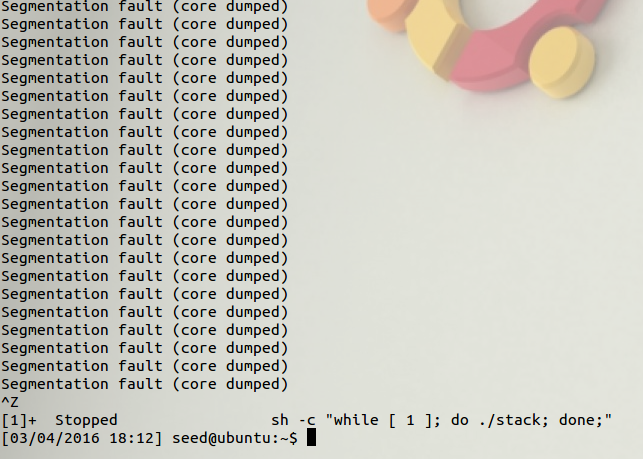
The shellcode acts as the other process we are attempting to execute on the machine we are exploiting. By pushing nonsense code onto the stack until we reach the stack pointer and replace the content of certain addresses, we can theoretically tell the system to perform just about any function we want. This includes gaining root access.



After filling up the buffer with enough null operations to reach the stack pointer, the shellcode is added to the “badfile.” Then badfile is pushed onto the stack generated by the stack.c program. Since we turned off address randomization we know where the address of the stack pointer and exactly how many *nop* instuctions to push. The shellcode allows us to access the root user.

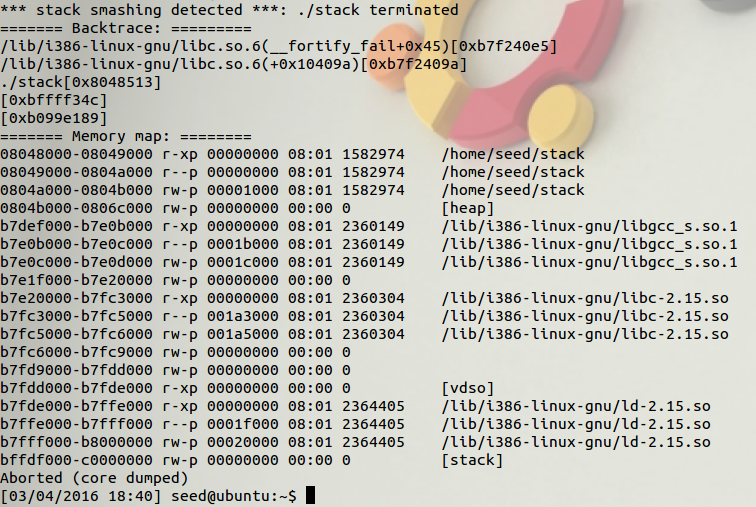


When the address randomization is turned back on, the code has virtually no success. Although it is possible for the system to stumble upon the shellcode it seems unlikely to do so and less likely to call the first set of instructions in shellcode.

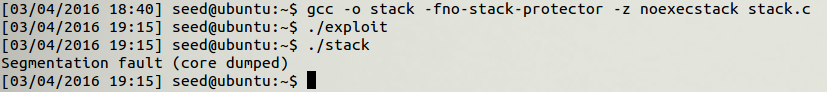


And since the shellcode is not in the expected address the system output is simply memory segmentation faults.

When the stack guard feature is implemented the potential overflow problem is reported to the user as well as the list of addresses affected. The program is terminated and an abort fault appears in the last line of the stack smashing prompt.



When the *noexecstack* command is used the stack is rendered “not-executable” which in other words, while the code is running the stack may be determined by the system to be inaccessible for some reason. Thus, a segmentation fault is seen since we decided initially the stack is not-executable.



Since the operating system is has certain fail safes for the stack already, it seems rather difficult to exploit the buffer overflow vulnerability without some initial privileges. However if an attacker manages to hit a critical memory location, it can give them not just sensitive information but possibly control of the user’s machine.