CS458 a1-milestone

Name: Poon, Darren Yuk Ho
Userid: dyhpoon
UWID: 20311433

sploit1.c

The first sploit is buffer overflow. In /usr/local/bin/backup, the copyFile() subroutine will copy src(source file) into the buffer with size of 4096, then it copies from buffer to dst(destination file). However, the copy process does not have any bound checking, and we could overwrite the buffer by supplying input that is larger than 4096bytes and rewrite the return address.

To be more specific, we could first fill our buffer with some "\x90" (NOP) instruction, then our shellcode. After that, we calculate and find the return address. We overwite this return address to point to any of our NOP instructions, so when the function returns, it will skip through to the shellcode and execute it, then we gain root access.

In order to fix this vulnerability, we have to make sure to do the bound checking when copying files into our buffer, make sure it does not overflow, so that the return address could not be changed.

sploit2.c

The second sploit is format string vulnerability. If we supply bad arguments to /usr/local/bin/backup, then the usage() subroutine is called. However, there is a vulnerability under the usage(), which is "printf(buffer); ". The user can supply the input(format string) directly to the printf() statement. In other words, the attackers could use this vulnerability to overwrite memories in the buffer, and execute shellcodes when printf() is return.

First of all, the usage() takes argv[0] as a parameter. We could change argv[0] by creating directories and using soft link to link it to /usr/local/bin/backup. Under the usage() routine, argv[0] is copied into the buffer, but buffer overflow does not work in this case because bound checking is performed under the snprintf() statement. However, we could still exploit through pure format string. We could overwrite

arbitrary memories to point the address to our shellcode by providing parameters and addresses, so when printf is return the shellcode will be executed. Now, the behavior of the format function is changed, and the attacker may get control over the application.

In order to fix this vulnerability, we should use printf() statement properly. We should use printf(“%s, buffer) instead of printf(buffer), so that the attacker is unable to provide the format string, and the behavior of the format function could not be changed.