Milestone/Exploit 1

a) The identified vulnerability is a buffer overflow in the *print_usage* function. This buffer overflow exists with the *strcat* function in the first part of the if statement. You can easily reach this *strcat* by running the *pwgen* program with no commands/flags. The buffer overflow occurs because *strcat* is copying $arg[0]$. The overflow does not occur naturally. To create the buffer overflow you must pass a large string into $arg[0]$ using *execve*.

b) My program exploits the vulnerability by taking advantage of the buffer overflow. I run the *pwgen* program using *execve* in my *sploit*1 program. With *execve* we can replace $arg[0]$. I replaced $arg[0]$ with a series of NOPS, shellcode, then a return address, in that order. This is all one large string that I pass as $arg[0]$. An example of this would be:

$$NOPNOPNOP...SHELLCODE...ADDRADDR$$

I have multiple return addresses in order to maximize the chances of pointer pointing to my malicious return address. My malicious return address points to an address within the NOPs and the NOPs then create a NOP sled. The NOP sled will keep going to the next NOP untill it reaches my SHELLCODE. Once the program reaches my SHELLCODE it will create a shell, and because of the environment of pwgen the shell has root access.

c) You could do multiple things to fix this vulnerability

   (a) Replace the $arg[0]$ with a static "pwgen". We know the program being run will be pwgen, so we can just make the output contain "pwgen" rather than letting the user possibly decide $arg[0]$.

   (b) Have a check that checks the size of $arg[0]$. The buffer is 512 bytes long, and it is starting with x bytes because of the *strcpy*. So you could check if the size of $arg[0]$ is more than $512 - x$.

   (c) Use *strncat*. While this may be problematic in other ways because it truncates the string, this will specifically prevent a buffer overflow attack.