-First we entered the output of shell code into ‘egg’ script such that an environment variable named ‘egg’ will be initialized when ‘invoke -e egg=$(./egg) agent-brown $(./arg)’ executes.

-After that, we know that we can only overflow the buffer by one byte since the ‘for’ loop in agent-brown.c allows ‘i’ to be <=24, which means we can insert 25 characters in to ‘buf’ since ‘i’ starts from 0, while only 24 bytes is allocated for ‘buf’

-Also, every byte entered into the buffer has to xor with an unsigned 00100000.

-We ran ‘invoke -d -e egg=$(./egg) agent-brown $(./arg)’ to get into debugging mode and set a breakpoint at invoke.   
-We used ‘info register ebp’ to get the address of ebp while invoke() is executed, which is 0xbffffab8

-We also used ‘print &buf’ to get the address of the character array, which is 0xbffffa78.

-Thus we know that ‘buf’ is 64 bytes below ebp and we can insert 65 characters, so we are able to overflow and modify the last byte of the ebp

-Since we need the ebp to point back to inside the buffer, we can change the last byte of the ebp by overflowing ‘buf’.

-The last byte of the starting address of ‘buf’ is 78, which is ‘x’ in ascii. However, the inputs will be xor with 00100000 so we have to insert ‘X’ as the last char in the buffer.

-Therefore, we insert 64 bytes of ‘junk’ (we’ll specify what we entered for the first 64 bytes later on), and for the 65th byte, we input ‘X’.

-That way, the ebp will be overflowed and contains the address 0xbffffa78. When ‘leave’ is executed, it will pop %ebp and ebp will now point to buf[0].

-When the next method, dispatch, completes, mov %ebp,%esp will move the %esp to %ebp. And when it pops %ebp, %esp will now point to buf[1].

-Therefore, we have to store the inverse address of the beginning of the shell code in that block.

-By executing ‘x/s \*((char \*\*)environ+1)’, the address of the environment variable, egg, is 0xbfffff96.

-However, the shell code starts 4 bytes after that, which is 0xbfffff9a. We can see that by executing ‘x/32xw 0xbfffff96’. It will list out the values stored within ‘egg’ and we can see that the actual shell code begins at 0xbfffff9a.

-Since flip() will flip the bytes of the inputs, we inserted the flipped version of 0xbfffff9a for the first 64 bytes, for simplicity. The flipped version of 0xbfffff9a is 0x9fdfdfba. Therefore, we filled up the buffer with \xba\xdf\xdf\x9f for the first 64 bytes and “X” as the 65th byte.

-Therefore, when we run ./exploit, the esp will eventually point to the buffer and cause the eip to point to the beginning of the shell code and spawns the shell script.