Pico CTFs Write Ups week 3:

1. ASCII Numbers:
    a. Used an ASCII table to convert the list of hexadecimals to a string and get the flag.
    b. picoCTF{45c11_n0_qu35710n5_1ll_t311_y3_n0_l135_445d4180}
2. Picker 1:
    a. Downloaded the python script and used the command "cat picker-l.py" to read the code.
    b. In the code there was a function called “win” that only worked if you had a file named "flag.txt" in the same directory as the script and it would give you the flag in hexadecimal numbers.
    c. Used the command "nc saturn.picoctf.net 50513" to connect and run the program and instead of writing "getRandomNumber" like the problem said, I wrote"win" as to call the function mentioned before and it worked.
    d. It gave a list of hexadecimal numbers that I converted to text with an ASCII table, and it gave me the flag.
    e. picoCTF{4_d14m0nd_1n_7h3_r0ugh_ce4b5d5b}
3. Bit-O-Asm-1
    a. Downloaded the assembly dump and saw that a “mov” statement moved 0x30 to the target register, in decimal this is 48.
    b. The flag was picoCTF{48}
4. Bit-O-Asm-2
    a. Downloaded the dump and found that inside the target register there was a pointer called "DWORD" that had the value 0x9fe1a.
    b. In decimal the value is 654874 so the flag was picoCTF{654874}
5. Bit-O-Asm-3
    a. Downloaded the assembly dump and followed the target register.
    b. After an “imul” and “add” instruction the resulting number inside the register was 0x27FA5D or 2619997 in decimal.
    c. The flag was picoCTF{2619997}
6. Bit-O-Asm-4
    a. Downloaded the assembly dump and followed the target register.
    b. In the dump a comparison was made to see if the value on the target register was less than a given number, but it failed and the “jcc” wasn't needed.
    c. After that a subtraction was made and then a “jmp”, as a result the number on the register was 654773.
    d. The flag was picoCTF{654773}
7. GDB baby step 1
    a. Downloaded the program to debug and used the command "gdb debugger0_a" and then "(gdb) disassemble main" to get a dump of the main function.
    b. Afterward, I followed the target register and at the end it had the number 549698.
    c. The flag was picoCTF{549698}

8. GDB baby step 2
    a. Downloaded the program to debug and used the command "chmod +x debugger0_b" to make it executable.
    b. Used the commands "gdb debugger0_b" and "(gdb) disassemble main" to get the dump.
    c. Since it had a loop, the problem recommended using "break *main+59", "run" and "info registers eax" to get the final value of the register which was 307019.
    d. The flag was picoCTF{307019}
9. GDB baby step 3
    a. Downloaded the program to debug and used the commands "gdb debugger0_c" and "(gdb) disassemble main" to get the dump.
    b. Since the point of the problem was to learn how to see values in memory I used "break *main+25" to break right after the value 0x2262c96b was loaded into memory, "run" and "x/4xb $rbp-0x4" to examine and get the value but this time in reverse since it was in little endian.
    c. The flag was picoCTF{0x6bc96222}
10. PW Crack 1
    a. Downloaded the python program and encrypted flag.
    b. Used the command "python3 level1.py" and got prompted for a password which it was unknown.
    c. After that I used the command "cat level1.py" to read the code and in an if statement at the bottom of the code was the password "1e1a"
    d. Finally used the command "python3 level1.py" again, used the password and got the flag.
    e. picoCTF{545h_r1ng1ng_fa343060}
11. PW Crack 2
    a. Downloaded the python program and the encoded file.
    b. Used the command "nano level2.py" to read the code and found that it asked for a password like level1, and it was in an if statement but this time in hexadecimal, so I had to use an ASCII table if I wanted to convert it.
    c. To skip that process I just deleted the input line and pasted the password directly from the if statement and then used the command "python3 level2.py" and got the flag.
    d. picoCTF{tr45h_51ng1ng_489dea9a}
12. PW Crack 3
    a. Downloaded the python program, encoded flag, and the hash file.
    b. Used the command "cat level3.py" to read the code and it uses hashes to compare the passwords. It reads the hash value of the correct password and compares it with the password that the user inputs after passing it through a hashing function "hashlib.md5()", if its correct it returns the flag.
    c. Therefore, I used the command "bvi level3.hash.bin" and got the hexadecimal values "e16d55a55d80dddd52a83eabea572b7b", I sent these values to an online

hash md5 decoder and got the password "87ab" and this string was in the possible passwords given by the problem.

d. Finally, I used the command "python3 level3.py", entered the password and got the flag.
e. picoCTF{m45h_fl1ng1ng_cd6ed2eb}