1. Write a C program which take into account two integers A, B and output the greatest common divisor of both A and B. Example: A = 10, B = 15 then output 5; A = 7, B=19 then output 1.

**The program prompts the user to input the dividend (integer A) and divisor (integer B) and prints the result. If divisor entered is zero, the program re-prompts the user to enter a new divisor. The program displays the computed greatest common divisor as a positive integer even if the one or both of the integers entered are negative. File name: Snyder\_HW\_01\_01.c**

1. Write a C program which asks the user for an integer N. Output the prime number P that is closet to N and is greater than N. Example: 10 then output P = 11; N = 23 then output P=29.

**The program … <>.**

1. Write a C program that allows the user to enter two integers and output their sum and product. Observe the behavior of your program when the two integers were big.

**When both integers entered were large … <>.**

1. Write a “*Hello World*” C program with some extra initialized and uninitialized variables. This program will also be used in questions 7, 8, 9, and 10. When you compile it using gcc
2. What is the effect of the option *–Wall*?

**The ‘-Wall’ option enabled all warnings about constructions and language-specific warnings. In the case of the “Hello World” program I created, it specifically warned about the four unused variables I had in the program (‘tChar’, ‘fNum2’, ‘Fnum1’, and ‘dNum’). File name: Snyder\_HW\_01\_04\_a.txt**

1. What is the effect of the option *–pedantic*?

**The ‘-pedantic’ option will issue all warnings demanded by strict ISO C and C++; rejecting all programs not meeting the ISO C or ISO C++ or using forbidden extensions. This makes the code easier to compile with other compilers that implement the same standard. In the case of the “Hello World” program I created, no warnings were issued. File name: Snyder\_HW\_01\_04\_b.txt**

1. What is the difference in the size of the code compiled with and without *–g*?

**The size of the compiled code with the ‘-g’ option is larger than the code compiled without the ‘-g’ option. In the case of the “Hello World” program I created, the size of the code compiled without the ‘-g’ option was 7217 and with the ‘-g’ options was 8329. File name: Snyder\_HW\_01\_04\_c.txt**

1. What is the difference in the size of the code compiled with *–O1*, *–O2*, and *–O3*?

**The size of the code compiled will be optimized with the ‘-O1’ option, will be optimized even more with the ‘-O2’ option, and again even more than the other two options with the ‘O3’ option. In the case of the “Hello World” program I created, the size of the code compiled with the ‘-O1’ option was 7217, with the ‘-O2’ option was 7217, and with the ‘-O3’ option was 7217.**

1. Open the C program in the debugger *gdb*. Set a breakpoint at the first line. Run the program to that point.
2. What is the current *ESP*?

**esp 0xbffff320 0xbffff320**

1. What is the current *EBP*?

**ebp 0xffff358 0xffff358**

1. What is the current *EIP*?

**eip 0x804843d 0x804843d <main+9>**

1. Use the *x* command to show the assembly language code for the next few commands to be executed.

**=> 0x804843d <main+9>: mov %gs:0x14,%eax**

**0x8048443 <main+15>: mov %eax,0x2c(%esp)**

**0x8048447 <main+19>: xor %eax,%eax**

**0x8048449 <main+21>: movl $0x0,0x18(%esp)**

**0x8048451 <main+29>: movb $0x21,0x1f(%esp)**

1. Use the *strace* command to list all of the system calls that your program makes.

**In the case of the “Hello World” program, the unique system calls used are execve, brk, access, mmap2, open, fstat64, close, read, set\_thread\_area, mprotect, munmap, write, and exit\_group. Many of the system calls appear multiple times. File name: Snyder\_HW\_01\_05.txt**

1. Use *objdump* to find the address of the following sections in your program:
2. *.text*

**Idx Name Size VMA LMA File off Algn**

**0 .text 0000005e 00000000 00000000 00000034 2\*\*2**

1. *.bss*

**Idx Name Size VMA LMA File off Algn**

**2 .bss 00000000 00000000 00000000 00000094 2\*\*2**

1. *.data*

**Idx Name Size VMA LMA File off Algn**

**1 .data 00000000 00000000 00000000 00000094 2\*\*2**

**File name: Snyder\_HW\_01\_09.txt**

1. Rewrite the “*helloASM.*asm” program described in class. Compile it and link it.
2. Include the source, object, and executable code.
3. Debug the program. What memory addresses contain the program’s environment strings?
4. Debug the program. What memory addresses contain the program’s arguments?
5. Debug the program, and stop it immediately before the syscall for *write*.
   * What is the current *ESP*?
   * What is the current *EBP*?
   * What is the current *EIP*?
6. Debug the program. Where is the message string located?
7. What is the return value for the *write* syscall? Where is it located?
8. Write a C program that contains a stack based buffer overflow. Explain in detail why the program has a stack based buffer overflow flaw. Demonstrate the flaw by causing the program to crash with a segmentation fault. Include the state of the stack before the crash, and determine exactly why the program crashed.
9. Write a C program that contains a stack based buffer overflow, and make it SUID root. Run the program outside of the debugger as an unprivileged user, and exploit the overflow to obtain a root shell. Include a description of exactly how the program was exploited, and a screen shot showing a root shell was obtained.