CSC 438 Systems and Software Security, Spring 2014
Instructor: Dr. Natarajan Meghanathan
Question Bank for Module 6: Software Security Attacks

1) What will be the output of the following C program when compiled and executed? Explain the results that you obtain:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){

    int diff, size = 8;
    char *buf1, *buf2;
    buf1 = (char*) malloc(size);
    buf2 = (char*) malloc(size);

    diff = buf2 - buf1;

    memset(buf2, '2', size);
    printf("BEFORE: buf2 = %s \n", buf2);

    memset(buf1, '1', diff + 3);
    printf("AFTER: buf2 = %s \n", buf2);

    return 0;

}
```

2) Consider the following C code. Assume the following: (i) architecture used is Little-endian; (ii) 32-bit memory addresses and a 4-byte stack boundary; (iii) the largest assigned high end memory address of the stack is 0x8abf4d40. Use this as the starting address to show the structure of your stack frames.
For (c) and (d), do not need to store the characters of the user inputs in their ASCII values in the stack. Just show them as characters itself.

```
void foo ( ){
   char c[8];
   gets(c);
   puts(c);
}

int main( ){
    foo( );
}
```

(a) Show the structure of the stack before the call to the foo( ) function.

(b) Show the updated structure of the stack after the call to the foo( ) function.

(c) Assume the user passes a 7-byte character input "JACKSON" using the gets( ) function. Show the structure of the stack before and after executing the gets( ) function in foo.

(d) Assume the user passes the string “ATTACKERCODE\x8C\35\xC0\x90” as input using the gets( ) function. Show the structure of the stack before and after executing the gets( ) function in foo.

(e) If you were to pass the input for part (d) from command line, how would you pass the input. Assume the name of the executable is *demo*.

3) Consider the following C code. Assume the architecture used is Big-endian. Assume the largest assigned high end memory address of the stack is 0x8abf4d40. Use this as the starting address to show the structure of your stack frames. There is no need to store the characters of the user inputs in their ASCII values in the stack. Just show them as characters itself.

```
int main(int argc, char *argv[ ]) {
        int valid = FALSE;
        char str1[8];
        char str2[8];

        str1 = "JACKSON";
        gets(str2);

        if (strncmp(str1, str2, 8)) == 0)
              valid = TRUE;

        printf("buffer: str1(%s), str2(%s), valid (%d)\n", str1, str2, valid);

}
```

(a) Show the structure of the stack frame for the main( ) function before the gets( ) call is executed.

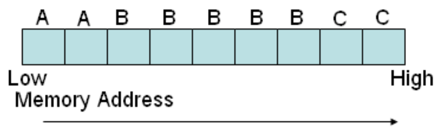
(b) Show the structure of the stack frame for the main( ) function after the user passes "JACKSON" as input the gets( ) function, and also write the values that will be printed out based on the printf( ) function.

(c) Show the structure of the stack frame for the main( ) function after the user passes "JACKSONSTATE" as input the gets( ) function, and also write the values that will be printed out based on the printf( ) function.

(d) Show the structure of the stack frame for the main( ) function after the user passes "COMPUTERCOMPUTER" as input the gets( ) function, and also write the values that will be printed out based on the printf( ) function.

4) Explain the idea of "NOP sledding" used to launch a buffer overflow attack.

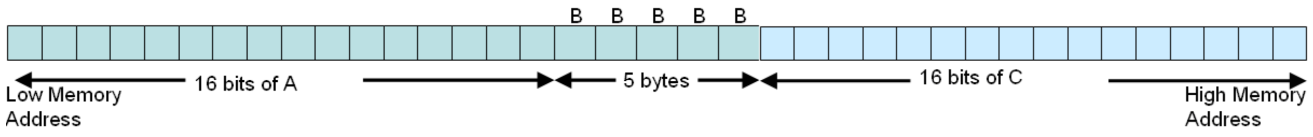
5) Consider the following layout for the memory: Assuming A, B and C are declared respectively as 2-byte integer, 5-byte character array and 2-bye integer. Let the initial values of A, B and C are respectively as follows: A = 4312, B = “0000”, C = 1816

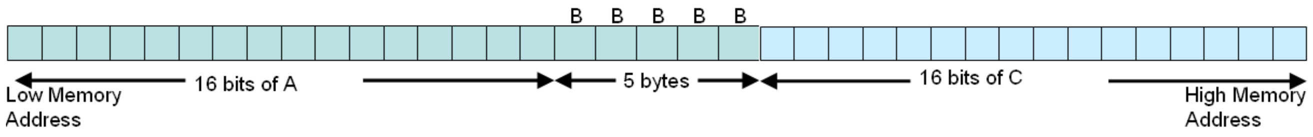


Note that the last character of a string is the terminating character ‘\0’ and assume its ASCII value is 0. Assume the ASCII value of ‘R’ to be 82.

Find the value of the integers A and C after we set B = “RIVER”, assuming the processor used is:
(i) BIG-ENDIAN

Initial Values for A = 4312, C = 1816

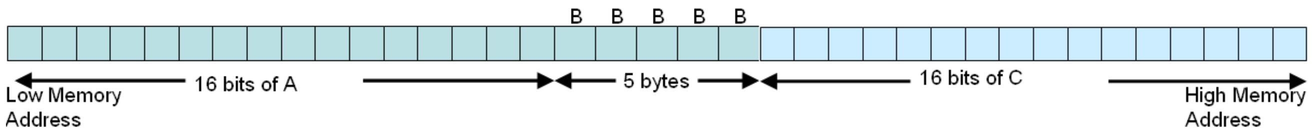


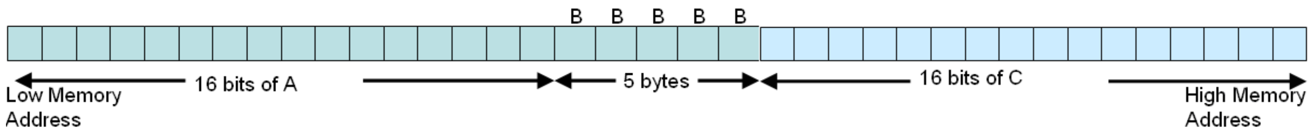


Final Values for A = _______________, C = _______________

(ii) LITTLE-ENDIAN

Initial Values for A = 4312, C = 1816





Final Values for A = _______________, C = _______________

6) Briefly explain how a stack smashing buffer overflow attack could be launched through the web?

7) In addition to use languages that have tight boundary checking mechanisms, briefly explain three strategies discussed in the lecture to detect or avoid stack smashing buffer overflow attacks.

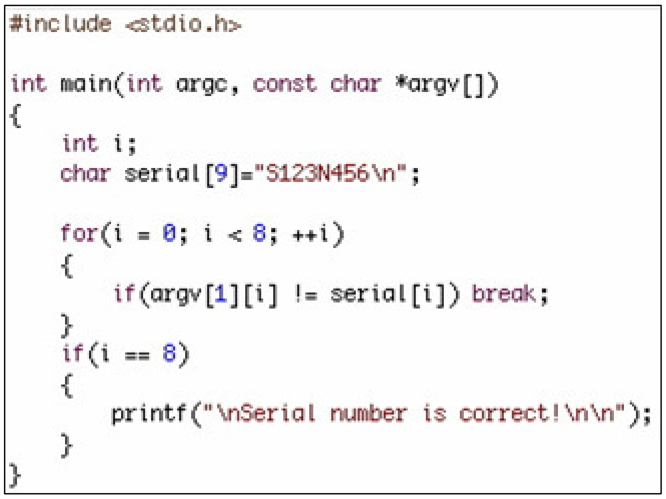
8) Give four examples of dependency-based attacks on application software.

9) What vulnerability a “try and buy” software faces with regards to validating whether a user has legitimately purchased the software?

10) What is “stress testing?” What kind of resources is normally targeted – how and why?

11) Briefly explain the cause of linearization attacks and how it is cleverly exploited by an attacker?

12)



```
#include <stdio.h>

int main(int argc, const char *argv[])
{
    int i;
    char serial[9]="S123N456\n";

    for(i = 0; i < 8; ++i)
    {
        if(argv[1][i] != serial[i]) break;
    }
    if(i == 8)
    {
        printf("\nSerial number is correct!\n\n");
    }
}
```

Consider the C code (to the left) and explain how an attack could be launched to determine the serial number hard coded in the program in linear time rather than exponential time. The set of symbols is {A-Z, a-z, 0-9}. Assuming that you start with a serial number AAAAAAAA, and try the symbols in the order A - Z, a - z, followed by 0 - 9, what would be the number of iterations the program has to be run to crack the code?

13)

```
#include <stdio.h>

int main(int argc, char * argv[])
{
  unsigned int connections = 0;
  // Insert network code here
  // ...
  // ...
  // Does nothing to check overflow conditions
  connections++;
  if(connections < 5)
    grant_access();
  else
    deny_access();
  return 1;
}
```

Consider the C code snippet (to the left) that is executed to get access to the Internet as part of a larger application program. The application program prefers to have only at most 4 Internet connections at any point of time and the C code snippet has been programmed for that purpose. Analyze the C code and describe the potential vulnerability that could be exploited by an attacker to get more than 4 Internet connections? After the application runs 4 Internet connections, how many times the C code has to be executed to get the 5th Internet connection? Assume a 4-bit architecture system.

14) Briefly explain the strategy through which a format string attack is launched.

15) Consider the C code below: Answer what would be the estimated output or the exact output (depending on the case) when the argv[1] to the *printf* function is passed with the following different values:

   a. if argv[1] is passed "%08x %08x %08x %08x %08x\n"
   b. if argv[1] is passed "\x10\x01\x48\x08 %s"
   c. if the printf function in the above code was called like this: printf(argv[1], &i) where i is an integer variable, and the value of argv[1] is passed "12345%n",

```
#include <stdio.h>
int main(int argc, char * argv[])
{
  printf("Your argument is:\n");
  // Does not specify a format string, allowing the user to supply one
  printf(argv[1]);
}
```

16) Consider the C code snippets shown to the left and the right. Both are file reader programs with the *setuid* bit set. The code on the right side is the solution to prevent a TOCTTOU attack that may be launched on the code at the left side. Explain:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <fcntl.h>
int main(int argc, char * argv[])
{
  int file;
  char buf[1024];
  memset(buf, 0, 1024);
  if(argc < 2) {
    printf("Usage: printer [filename]\n");
    exit(-1);
  }
  if(access(argv[1], R_OK) != 0) {
    printf("Cannot access file.\n");
    exit(-1);
  }
  file = open(argv[1], O_RDONLY);
  read(file, buf, 1023);
        close(file);
  printf("%s\n", buf);
  return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <fcntl.h>
int main(int argc, char * argv[])
{
  int file;
  char buf[1024];
  uid_t uid, euid;
  memset(buf, 0, 1024);
  if(argc < 2) {
    printf("Usage: printer [filename]\n");
    exit(-1);
  }
  euid = geteuid();
  uid = getuid();
  seteuid(uid);
  file = open(argv[1], O_RDONLY);
  read(file, buf, 1023);
  close(file);
  seteuid(euid);
  printf("%s\n", buf);
  return 0;
}
```

a. How could one launch a TOCTTOU attack on the code at the left side? What is the vulnerability that is exploited?

b. How does the code on the right side fix the vulnerability of the code at the left side? What is the main software coding principle adopted to fix the vulnerability?