**UNIVERSITY OF HERTFORDSHIRE**

Academic Year: 2018/2019

Semester: B

School of Computer Science

**7COM1028**

**Secure Systems Programming**

**Referral Coursework: Secure Programming**

**(50%)**

Please fill in your student ID

**Student ID \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(**staff use only**)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| Total |  |  |  |

There are 5 questions of 10 marks each.

Full marks may be obtained by attempting ALL questions.

Question1:

The following program contains five errors. Identify the errors and fix them. (10 marks)

|  |
| --- |
| 1. int main(void) { 2. char source[10]; 3. char \*dest; 4. size\_ti; 5. strcpy(source, "0123456789"); 6. dest = malloc(strlen(source)); 7. for (i = 1; i <= 11; i++) { 8. dest[i] = source [i]; 9. } 10. dest[i] = '\0'; 11. printf("dest = %s", dest); 12. return 0; 13. } |

Answer:

In line 5 source size is 10 bytes long but we tries to copy 11 bytes , this error can be fixed by increasing the size of source variable to 11.

In line 6, strlen will not count the null byte this will allocate only 10 bytes and it can be fixed by using sizeof in place of strlen.

In line 7, i starts from 1 which is wrong as it should starts from 0.

In line 9, i terminates with value 11 but it should only loop through 10. It can be fixed by changing the terminating condition.

In line 10, i value will be 11 which is wrong as it can have maximum value of 10. It can be fixed by fixing the line 9.

Question 2

The following program is for Question 2.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>   1. int main(int argc, char \*argv[]) { 2. char \*first, \*second, \*third; 3. if (argc< 2) 4. {    1. printf("usage: ./q1 str\n");    2. exit(1); 5. } 6. first = (char \*) malloc(660); 7. second = (char \*) malloc(220); 8. third = (char \*) malloc(120); 9. strcpy(second, argv[1]); 10. printf("locations of the 1st, 2nd, and 3rd memory trunks are %p, %p, %p \n", first, second, third); 11. printf("contents of the 2nd memory trunk are %s\n", second); 12. free(first); 13. free(second); 14. free(third); 15. } |

In the program, %p in the printf() function prints the location of the allocated memory chunk. Compile the program to build the executable file.

1. Illustrate how the 1st, 2nd and 3rdchunks of memory are allocated and linked in the program. Work out the number of bytes needed for each memory chunk and the values in the memory chunk header.

Answer: In the two runs of the program output are as follows:

locations of the 1st, 2nd, and 3rd memory trunks are 0000000000A51400, 0000000000A516A0, 0000000000A51790

locations of the 1st, 2nd, and 3rd memory trunks are 0000000000AE1470, 0000000000AE1710, 0000000000AE1800

Both runs shows that the memory are allocated in sequential manner.

Number of bytes for first is 664 bytes, for second is 224 bytes and for third is 124 bytes.

1. Analyze the vulnerability of the program to a buffer overflow attack by using a malicious argument of argv[1] which contians 224 bytes with the last four bytes being written with the value 4.

Answer: Buffer will overwrites the size of the third memory chunk to -4 which sets the bit of the third memory chunk to 0.

Question 3

The following programis for Question 3.

|  |
| --- |
| #include <stdlib.h>  #include <string.h>  #include <stdio.h>   1. void getComment(unsigned int len, char \*src) { 2. unsigned int size; 3. size = len – 4; 4. char \*comment = malloc(size+1); 5. memcpy(comment, src, size); 6. printf("%s\n", comment); 7. return; 8. } 9. int main(int argc, char\* argv[]) { 10. if (argc< 3) { 11. printf("Usage: %s <str1><str2>\n", argv[0]); 12. return -1; 13. } 14. getComment(atoi(argv[1]), argv[2]); 15. return 0; 16. } |

Compile the program to build the executable file.

1. Give an example of a normal run of the program. Illustrate clearly the input, output and the values of the variables “len” and “size” in your example.

Answer:Below is the given example of the normal run of the program.

Input which is given while executing on the terminal or on console in window:

**Input**: a.exe 10 abcd

Output obtained is:

**Output**: abcd

**And the value of :**

Value of len = 10

Value of size = 6

1. Give an example of an abnormal run of the program which ends up with “Segmentation Fault” Analyze what causes the problem and how the vulnerability can be fixed. Fix the program and run it again with the same input. The “Segmentation Fault” should not happen again.

Answer: Segmentation fault occur when argv[1] value is less than the size of the length of the argv[2].It occurs when the program crash and it is when our program tries to read or write any illegal memory locations. And this same happens in our program when first argrument size is less than the size of the second argument. As in that case it tries to access those memory locations which is not permissible to access.

This problem can be fixed by adjusting the value of size as because of this it tends to access some unauthorized memory locations.It can be fixed by changing the line unsigned int size = line – 4; , to unsigned int size = line. By doing above solution it can be fixed.

Question 4

The following program is for Question 4.

|  |
| --- |
| #include <stdio.h>  #include <string.h>   1. int main() { 2. char aaddr[9], format[30], buffer[30]; 3. int aaa = 16; 4. int bbb = 0; 5. sprintf(aaddr, "%08x", &aaa); 6. printf(aaddr); 7. printf("\n"); 8. sprintf(buffer, "\\x%c%c\\x%c%c\\x%c%c\\x%c%c", aaddr[6], aaddr[7], aaddr[4], aaddr[5], aaddr[2], aaddr[3], aaddr[0],aaddr[1]); 9. strcpy(format, buffer); 10. sprintf(buffer,"%%08x.%%08x%%n"); 11. strcat(format, buffer); 12. printf("%s\n", format); 13. printf(format, &aaa, &aaa, &bbb); 14. } |

Compile the program to build the executable file.

1. Demonstrate your understanding of the program by adding comments to the program from line 2 to line 13.

Answer:

// create three character array

char aaddr[9], format[30], buffer[30];

// create an integer variable and initialise it with value 16

int aaa = 16;

// create an integer variable and initialise it with value 0

int bbb = 0;

// store output in aaddr(character buffer)

sprintf(aaddr, "%08x", &aaa);

// print the character buffer

printf(aaddr);

// print new line

printf("\n");

// store the output in buffer(character buffer)

sprintf(buffer, "\\x%c%c\\x%c%c\\x%c%c\\x%c%c", aaddr[6],aaddr[7], aaddr[4], aaddr[5], aaddr[2], aaddr[3], aaddr[0],aaddr[1]);

// copy content of buffer in format array

strcpy(format, buffer);

// store the output in buffer(character buffer)

sprintf(buffer,"%%08x.%%08x%%n");

// append buffer array to the format array

strcat(format, buffer);

// print the format array

printf("%s\n", format);

// print the format array along with address of aaa and bbb

printf(format, &aaa, &aaa, &bbb);

1. In the program, an attacker changes line 13 to

13printf(format);

What is the attacker trying to do?What are the consequences?

Answer: By changing the line 13 to printf(format),attacker is trying to overwrite the memory. Because of this the previously stored value will be changed and this led to the unexpected run of the progam, which is not accepted. Consequences of this is that the values of variables aaa and bbb will be changed from this statement.

Question 5

1. What is the least privilege principle in secure software development? How can it be enforced in a password-changing program where users must be able to modify their own passwords, but must not be given free access to read or modify the password file containing all user passwords?

Answer: The least privilege principle in secure software development means that for performing secure operations processed should execute with the minimum permission.

In the password-changing program only user would be able to give input and based on other checking only the entry for that particular user will be changed.

1. What is TOCTOU? In the above example, how to avoid TOCTOU race conditions that could occur during the I/O operations of the password file?

Answer: TOCTOU stands for time of check,time of use. It is a class of software bugs caused by the changes in the system between the checking of conditions and the use of the results of that checking process. To avoid the TOCTOU race conditions that could occur during the I/O operations of the password file, we need to combines the process of opening the file and checking for file permission into an atomic operation as in this case between these two processes TOCTOU could occur.

1. Give an example to show how an attacker can gain root access over the network by exploring buffer overflow vulnerabilities.

Answer: Attacker can gain root access over the network by exploiting a vulnerability in a set root uid program.