Yuvika Nibber

Assignment 6

1. Buffer overflow is when the program’s buffer adds more information than its allocated size.
2. Middle level and low-level programming languages.
3. Compile time defenses and run time defenses.
4. Return to system call attack is the process that replaces the return address of an overflowing stack. It uses stack overflow process. The return address of a stack is usually replaced by another institution. They overwrite the stack because it uses exceeded memory. It is used to avoid the non-executable stack limitation.
5. Heap is a data structure that stores the value as a tree structure. The heap buffer overflow attacks the location on the heap. The heap management points are attacked in this buffer. The application overwrites the pointers that are available in the heap and caused the data to be corrupted.
6. The buffer in the global data area overflow is programmed to attack the global data area of a program. The buffer is overwritten by using unsafe buffer operations which causes the pointers memory location. The overwritten function sends the control to the attacker who wrote the main program.
7. int main (int argc, char \*argv[ ]) {  
   int valid=FALSE;  
   char str1[8];  
   char str2[8];     
   next\_tag(str1);  
   fgets(str2, 8, STDIN);   
   if (strncmp(str1,str2,8)==0)     
   valid=TRUE;  
   printf("buffer: str1(%s),str2(%s),valid(%d) \n", str1, str2, valid);  
   }
8. void gctinp ( char \* inp , int siz )

{

puts (" Input value : ") ;

fgets ( inp , siz , stdin ) ;

printf (" buffer3 getinp read %s\n", inp ) ;

}

void display(char \*val)  
{  
char tmp[30];  
sprintf(tmp, "read val: %s\n", val);  
puts(tmp);  
}

int main ( int argc , char \* argv [])

{

char buf [16];

getinp ( buf , sizeof ( buf ) ) ;

display ( buf ) ;

printf (" buffer3 done \n") ;

}

1. int corrected\_copy\_buf(char \*to, int pos, **int size,** char \*from, int len){

int i;

**if(pos<0) {**

**pos = 0**

**}**

**if(pos+len > size) {**

**printf("\n provided value for 'len' is greater than the size of the destination array, text to be copied will be trucated.");**

**len = size - pos**

**}**

for(i=0;i<len;i++){

to[pos] = from[i];

pos++;

}

return pos;.

}

short corrected\_read\_chunk(FILE fil, char \*to, **short size**) {

short len;

fread(&len, 2, 1, fil);

**if(len>size){**

**printf("\nThe length specified is greater than the size of the array. Nothing written...");**

**return 0;**

**}**

fread(to, 1, len, fil);

return len;

}

1. #define INP\_SIZE 64

typedef struct chunk

{

char inp[INP\_SIZE];

void (\*process) (char \*);

} chunk\_t;

void showlen(char \*buf)

{ int len;

len = strlen(buf);

printf("buffer5 read %d chars\n", len);

}

int main(int argc, char \*argv[])

{

chunk\_t \*next;

setbuf(stdin, NULL);

next = malloc(sizeof(chunk\_t));

next->process = showlen;

printf("Enter value: ");

fgets(next->inp , INP\_SIZE,stdin);

next->process(next->inp);

printf("buffer5 done\n"); }