

**Computer Security**

**Tutorial 9**

**Part 1:**

Buffer overflows are a common type of programming vulnerability, particular for software written in variants of the C language. Below are some examples of buffer related problems. Examine the source code and try to follow its logic. See if you can find the buffer related problem in each example…

**Code Example 1:**

int main()

{

int buffer\_array[5] = {1,2,3,4,5};

printf(“%d\n”, buffer\_array[5]);

}

**Code Example 2:**

int main()

{

int buffer\_array[5];

int count;

for (count=0; count<=10; ++count)

{

buffer\_array[count] = count;

}

}

**Code Example 3:**

int main(char \*input)

{

char buffer[10];

strcpy(buffer, input);

}

**Part 2:**

Research the following terms/projects;

* Shellcode
* Opcodes
* Metasploit project
* Damn Vulnerable Web App

**Part 3: Advanced Question**

Consider the following piece of C code:

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

{

char continue = 0;

char password[8];

strcpy(password, argv[1]);

if (strcmp(password, "CS166")==0)

continue = 1;

if (continue)

{

\*login();

}

}

In the above code, \*login() is a pointer to the function login() (In C, one can declare pointers to functions which means that the call to the function is actually a memory address that indicates where the executable code of the function lies).

1. Is this code vulnerable to a buffer-overflow attack with reference to the variables password[] and continue? If yes, describe how an attacker can achieve this and give an ideal ordering of the memory cells (assume that the memory addresses increase from left to right) that correspond the variables password[] and continue of the code so that this attack can be avoided.
2. To fix the problem, a security expert suggests to remove the variable continue and simply use the comparison for login. Does this fix the vulnerability? What kind of new buffer overflow attack can be achieved in a multiuser system where the login() function is shared by a lot of users (both malicious and nonmalicious) and many users can try to log in at the same time? Assume for this question only (regardless of real systems' behavior) that the pointer is on the stack rather than in the data segment, or a shared memory segment.
3. What is the existing vulnerability when login() is not a pointer to the function code but terminates with a return() command? Note that the function strcpy does not check an array's length.