

CS2107 Tutorial 8 (Software Security)

School of Computing, NUS

26 – 30 October 2020

1. (*Buffer overflow vulnerability*): Try out this `badprogram.c` C program:

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

int main(int argc, char **argv)
{
    char text[16];

    strcpy(text, argv[1]); /* copy the 1st argument into array "text" */
    printf("The supplied first argument is :\n");
    printf("%s", text);
    printf("\n");

    return 0;
}
```

Compile the program above (e.g. `gcc -o badprogram badprogram.c`), and execute it. Notice that the program takes in an argument. By executing, for instance, `./badprogram 'hello world'`, the program `badprogram` will take in the string `'hello world'` as an argument, store the argument into the array `text`, and then print it.

Try running it with different arguments. What would happen if the argument is:

- (a) `two words`
 - (b) `'two words'`
 - (c) a very long string (say, longer than 32 characters).
2. (*UNIX commands*): Familiarize yourself with UNIX/Linux commands like `ls`, `cat`, `sh`, `echo`.
 3. (*Safe/unsafe functions*): Find out more about the following C library functions. Which usages should be avoided, and why?
 - (a) `strcat (dest, source);`
 - (b) `strncat (dest, source, n);`
 - (c) `memcpy (dest, source, n);`

- (d) `strncpy (dest, source, strlen(source));`
- (e) `sprintf (str, f);`
- (f) `printf ("Please key in your name: "); gets (str);`
- (g) `scanf ("%s", str);`
- (h) `scanf ("%20s", str);`

4. (*Memory initialization*): Consider the following C program.

```
#include <stdio.h>

int main()
{
    unsigned char a[10000];

    for (int i=0; i<10000; i++)
        printf ("%c", a[i]);

    return 0;
}
```

- (a) What would be the output? What is its implication to secure programming?
- (b) A possible preventive measure is to always initialize the array. What is the disadvantage of doing that?

5. (*Integer overflow vulnerability*): Consider the following C program.

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

int main()
{
    unsigned char a, total, secret; // Each of them is a 8-bit unsigned integer
    unsigned char str[256];         // str is an array of size 256
    a = 40;
    total = 0;
    secret = 11;

    printf ("Enter your name: ");
    scanf ("%255s", str);           // Read in a string of at most 255 characters

    total = a + strlen(str);

    if (total < 40) printf ("This is what the attacker wants to see: %d\n", secret);
    if (total >= 40) printf ("The attacker doesn't want to see this line.\n");
}
```

If the user follows the instruction and enters his/her name honestly, he/she will be unable to see the secret. Suppose you are the attacker, how would you cause the secret number to be displayed?

6. *Terminology:* CVE, Black list, White list, Black hat, White hat, Spamhaus, CERT, SingCert, SOC (Security Operation Center), SIEM.

— End of Tutorial —