# Homework 5

CS161 Computer Security, Spring 2008

Assigned 4/23/08

Due 5/05/08

1. **Worm Propagation**

   In lecture, we talked about ways of increasing the propagation rate of worms. In this problem, we'll examine the effects of decreasing the propagation rate of worms.

   Recall that $i(t)$ is the proportion of machines in a network that are infected by a worm at time $t$, $\beta$ is the contact rate, and $T$ is a constant of integration that fixes the time position of the incident. We'll use a Random Spread (a.k.a Susceptible-Infected) model for worm propagation and assume a network of tens of millions of susceptible machines.

   Please limit each answer to 1-3 sentences. You may also include graphs or tables if you like, though they are not necessary.

   (a) (4 points) If $\beta$ is 3.5 and $T$ is 15, at what time are 50 percent of the machines infected? At what time will 99% of all machines be infected? Hint: An easy way to work through this problem is to use Mathematica, Excel, or OpenOffice to generate or graph your results.

   2 points: The 50 percent time is roughly 15.

   2 points: The 99 percent time is roughly 17.

   (b) (4 points) If we are able to reduce the initial infection rate to 0.5, what is the 50 percent infection time? What is the time for 99% of all machines to be infected?

   2 points: The 50 percent time is the same as in part a.

   2 points: The 99 percent time is roughly 25.

(c) (4 points) Sometimes a worm is initially distributed to a **hitlist**, a set of hosts known to be vulnerable. Once the hosts on the hitlist are infected, these hosts scan randomly to continue spreading. Consider a hitlist that makes up one percent of all vulnerable hosts. Modify the formula for i(t) to take into account the speedup gained by the hitlist.

i(t+x) where x satisfies i(x) = 1 percent.

2. **Honeypots and Tarpits**

A tarpit is a honeypot that consumes as many of the adversary's resources as possible. For each of the following honeypots, describe a way we could turn the honeypot into a tarpit. Do not use the same answer for more than one part of this problem. Keep your answers short, no more than two or three sentences.

(a) (3 points) An FTP server with no password. Located on the FTP server is a file with a tempting name, such as corporate_secrets.txt

Returning a file of unlimited length, containing random numbers, OR putting false and misleading corporate secrets into the file, so that the attacker acts on the false information, wasting his time.

(b) (3 points) A host with many open ports.

Any method in which opening a port takes a very long time.

(c) (3 points) An open directory on a web server. The directory has a tempting name, such as /highly_proprietary_software/source_code.

Listing an unlimited list of files, all of which contain nothing useful. OR, place code that appears to truly be highly proprietary source code, but in reality is something crafted to mislead and confuse the attacker, such as some open source software.

(d) (3 points) A compromised SSH server containing industrial control software to accompany already-stolen steel mill blueprints.

A variety of answers were accepted. The best involved inserting a bug into the software such that if the attacker builds a steel mill using the blueprints, and runs it using this software, the steel mill will explode after being in operation a short while.

3. **Taint Analysis**

In this question, you are to perform a taint analysis on the following code.

```
float area_circle(float radius)
{
        float pi=3.14;
        return pi*radius*radius;
}

float area_square(float width)
{
        return width*width;
}

int main(char** argv, int argc)
{
         float radius_1 = 7.2;
         float area_1 = area_circle(radius_1);
         float radius_2 = read_float_from_keyboard();  //radius_2 is tainted
         float area_2 = area_circle(radius_2);         //area_2 is tainted.
         float summed_area = 0;
         summed_area += area_1;
         summed_area += area_2;                        //summed_area is taint

         int n = read_int_from_keyboard();             //n is tainted
         int fibonacci = 1;
         int i;
         for(i = n; i != 0; i--)                      //i is tainted
            {
            fibonacci *= i;                           //fibonacci is tainted
            }

         float (*pt2Area)(float);

         printf("Enter 'c' for circles, enter 's' for squares: ");
         char which_area = read_char_from_keyboard();  //which_area is tainte

         if (which_area == 'c')
             pt2Area = area_circle;
         if (which_area == 's')
             pt2Area = area_square;
```

```
char buf[42];
printf("Enter a size: \n");
gets(buf);                                          //everything on the st

float size = string_to_float(buf);                  //size is tainted
float area_3 = pt2Area(size);                       //area_3 is tainted (1
printf("Area is: %f \n", area_3);
printf("The ratio of area 3 to area 2 is: %f \n", area_3/area_2);

return 0;
```

Part b:

buffer overflow, 2 points

$which_area$ not c or s, leading to a bad function pointer, 2 points

$area_2$ taking a value of zero, leading to a divide by zero, 2 points

4. **Symbolic Execution**

Consider the following code.

```
void f(void)
{
int step = 0; int user_increment = 0;
int start = 1;
printf("Would you like to count by 1? (y/n)\n");
char choice = read_char_from_keyboard();

printf("Start counting at: \n");
start = read_char_from_keyboard();

if (choice == 'n')
   {
   user_increment = read_int_from_keyboard();
   if (user_increment > (100 - start))
       printf("WARNING: You may not experience many iterations.\n");
}
if (choice == 'y')
```

```
   {
   step = 1;
   if (user_increment > 100)
       printf("WARNING: You will not experience many iterations.\n");
}

step += user_increment;

/* Place assertion here */
assert(step != 0)

printf("Counting to 100, incrementing by %d. \n", step);
for(int i = start; i <= 100; i+=step)
    printf("Currently we are at %d. \n", i);
```

(a) (1 point) Write the assertion that must be true for correct execution.
1 point: Anything that ensures that step != 0.

(b) (11 points) Identify each path this code might take (up to the assertion). For each path, give the path predicate. Determine whether or not each path is feasible. For each feasible path, give an example of input that would cause this path to be executed. For each feasible path, write a symbolic expression that must be satisfied in order for the assertion to **fail**. Determine whether each symbolic expression is satisfiable. For each satisfiable expression, give an example of input that causes the assertion to fail.

We were looking for 9 paths, with the conditionals taking the following branches (- means not reached):

i. TTTT
ii. TTTF
iii. TTF-
iv. TFTT
v. TFTF
vi. TFF-
vii. F-TT
viii. F-TF
ix. F-F-

Here are the corresponding path predicates:

i. choice = n and choice = y and ... (infeasible)
ii. choice = n and choice = y and ... (infeasible)
iii. choice = n and user increment > (100 - start) ; fail if user increment = 0
iv. choice = n and choice = y and ... (infeasible)
v. choice = n and choice = y and ... (infeasible)
vi. choice = n and user increment <= (100 - start) ; fail if user increment = 0
vii. choice = y and user increment > 100 ; cannot fail
viii. choice = y and user increment <= 100 ; fail if user increment = -1;
ix. choice != y and choice != n