CSCI4730/6730 – Operating Systems

Project #4

Due date: 11:59pm, 12/08/2015

Description

In this project, you will gain the first-hand experience on exploiting buffer-overflow vulnerability. A buffer overflow occurs when a process tries to write data beyond the boundaries of pre-allocated fixed length buffers. Buffer overflows can be triggered by inputs that are designed to execute code, or alter the way the program operates. This may result in a breach of Operating System security. In this project, you will be given a simple program with a buffer-overflow vulnerability. Your task is to develop a scheme to exploit the vulnerability and finally to inject and execute an arbitrary shellcode.
**Use "nike" server for this project.

Part 1: Buffer Overflow - 100%

"bof.c" has a buffer overflow vulnerability. It reads an input from a command-line argument and copy into buffer. Because strcpy() does not check boundaries, buffer overflow might occur. Most Linux-based systems use **Address Space Layout randomization (ASLR)** to randomize the starting address of heap and stack. This makes guessing the address difficult. In this part, we disable these features using following commands:

```
$ setarch `uname -m` -R /bin/bash
```

Another protection mechanism to prevent buffer overflows is **StackGuard**. In the presence of this protection, exploiting the buffer overflow will be very difficult. We will disable this protection when we compile the vulnerable program using the gcc option `–fno-stack-protector`. (It is already in the provided Makefile)

The last protection mechanism in Linux is **Data Execution Prevention (DEP).** It prevents attacks that store executable instructions in a data area via a buffer overflow. We will disable this protection using the gcc option `–z execstack` (Already in the provided Makefile)

In this part, you will follow the step-by-step instructions in the following document to exploit the buffer overflow vulnerability. Note that, you might have different addresses and offsets for the buffer and registers.
https://www.exploit-db.com/docs/33698.pdf

We will use the following shellcode (83 bytes) to inject and execute in the vulnerable process. **Do not use** the shellcode in the document.

```
"\xeb\x3f\x5f\x80\x77\x0c\x41\x48\x31\xc0\x04\x02\x48\x31\xf6\x0f\
x05\x66\x81\xec\xff\x0f\x48\x8d\x34\x24\x48\x89\xc7\x48\x31\xd2\x6
6\xba\xff\x0f\x48\x31\xc0\x0f\x05\x48\x31\xff\x40\x80\xc7\x01\x48\
x89\xc2\x48\x31\xc0\x04\x01\x0f\x05\x48\x31\xc0\x04\x3c\x0f\x05\xe
8\xbc\xff\xff\xff\x2e\x2f\x73\x65\x63\x72\x65\x74\x2e\x74\x78\x74\
x41"
```

Submission:

1. Describe your experience (including a few screen shots) and explain all steps you have done to exploit the buffer overflow.
2. Turn on the ASLR and run the same attack you have developed. Can you get the same result? If not, what is the problem? Describe your observation and the problem in your report. Execute the following command to turn on ASLR:
   `$setarch `uname -m` /bin/bash`

Part 2: Address Randomization – Extra Credit 100%

In this part, you will turn on the Address Space Layout Randomization (ASLR) and exploit the buffer overflow vulnerability in **"bof2.c".** Besides the buffer overflow, this program has a format string vulnerability as well. You can use the format string vulnerability to get the buffer address. You will inject and execute the same shellcode as in part1.

Submission:

1. Document your experience (including a few screen shots) and explain all steps you have done to exploit the buffer overflow.

References:

1. Buffer overflow:
   - http://cecs.wright.edu/~tkprasad/courses/cs781/alephOne.html
   - https://www.owasp.org/index.php/Buffer_Overflows
2. ASLR: https://en.wikipedia.org/wiki/Address_space_layout_randomization
3. DEP: https://en.wikipedia.org/wiki/Data_Execution_Prevention
4. StackGuard: https://en.wikipedia.org/wiki/Buffer_overflow_protection
5. Format string vulnerability:
   https://www.owasp.org/index.php/Format_string_attack