Lab 18.2: More on setuid and Scripts

Suppose we have the following **C** program (./writeit.c) which attempts to overwrite a file in the current directory named `afile`:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>

int main(int argc, char *argv[])
{
        int fd, rc;
        char *buffer = "TESTING A WRITE";
        fd = open("./afile", O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
        rc = write(fd, buffer, strlen(buffer));
        printf("wrote %d bytes\n", rc);
        close(fd);
        exit(EXIT_SUCCESS);
}
```

If you are taking the online self-paced version of this course, the source code is available for download from your **Lab** screen.

If the program is called writeit.c, it can be compiled simply by doing:

```
$ make writeit
```

or equivalently

```
$ gcc -o writeit writeit.c
```

If (as a normal user) you try to run this program on a file owned by root you'll get

```
$ sudo touch afile
$ ./writeit

wrote -1 bytes
```

but if you run it as root:

```
$ sudo ./writeit

wrote 15 bytes
```

Thus, the root user was able to overwrite the file it owned, but a normal user could not.

Note that changing the owner of **writeit** to root does not help:


```
$ sudo chown root.root writeit
$ ./writeit

wrote -1 bytes
```

because it still will not let you clobber afile.

By setting the **setuid** bit you can make any normal user capable of doing it:

```
$ sudo chmod +s writeit
$ ./writeit

wrote 15 bytes
```

You may be asking, why didn't we just write a script to do such an operation, rather than to write and compile an executable program?

Under **Linux**, if you change the **setuid** on such an executable script, it won't do anything unless you actually change the **setuid** bit on the shell (such as **bash**) which would be a big mistake; anything running from then on would have escalated privilege!

