MCS 494 final
2004.12.09
D. J. Bernstein

Your answers must be based solely on your own knowledge and the information on these sheets. You are not permitted to use books, notes, or computers.

Do not hand in these sheets. Anything that you want to have graded must appear in the answer booklet. Make sure that your name is on the front of the booklet.

You may find the following functions useful: `symlink("target","newlink")` creates `newlink` as a symbolic link to `target`; `rename("foo","bar")` changes the existing file `foo` to be named `bar`, simultaneously removing any existing file `bar`; `lseek(4,0,0)` moves file descriptor 4's file position back to byte 0. Don't worry about including the proper `.h` files for these and other functions.

Problem 1. A user clicks on a URL inside a PDF document, not realizing that the URL was created maliciously. The PDF viewer calls

```
command = malloc(strlen(url) + 20);
if (!command) return -1;
sprintf(command,"firefox %s",url);
system(command);
free(command);
```

to start a browser. The browser displays a web page. The user closes the browser window. Later he is surprised to discover that many of his files have disappeared. What was the URL?

Problem 2. The system administrator arranges for the commands

```
print-new-configs > /tmp/new-configs
[ -s /tmp/new-configs ] && ( mail root < /tmp/new-configs )
rm /tmp/new-configs
```

to be run every night. A local user `joe` creates a file `evil.c` and runs

```
gcc -o evil evil.c
./evil
```

after which the system administrator's commands corrupt `/etc/passwd`. What were the contents of `evil.c`?

Problem 3. How can the security hole in Problem 2 be fixed? Write the new commands.

Problem 4. Sometimes, inside a setuid program, `main` calls a library function that uses an environment variable set by an attacker. The author of `main` does not realize that the library function is using the environment variable, and the author of the library function does not realize that the program is running setuid. Answer exactly *one* of the following two questions, clearly identifying which one you have chosen to answer: (A) How can the library function check whether it is running setuid? (B) How can `main` prevent library functions from seeing the attacker's environment?

Problem 5. The system administrator arranges for the command

```
find / -name '*.core' -mtime +7 -print | xargs rm -f
```

to be run every night. His goal is to remove all old files on the system named `*.core`. For example, the output of

```
find / -name '*.core' -mtime +7 -print
```

might be

```
/home/djb/konqueror.core
/tmp/z7.core
```

indicating that `/home/djb/konqueror.core` and `/tmp/z7.core` have not been modified in 7 days.

A local user `joe` creates a file `evil.c` and runs

```
gcc -o evil evil.c
./evil
```

after which the system administrator's command destroys `/etc/passwd`. What were the contents of `evil.c`?

Problem 6. The system administrator runs an `inetd-edit` program that reads the existing contents of `/etc/inetd.conf` into an `x` array in memory, makes certain changes to the `x` array, and then calls

```
fd = open("/etc/inetd.conf",O_WRONLY|O_TRUNC);
if (fd == -1) return -1;
write(fd,x,xlen);
close(fd);
```

to save the `x` array to disk. Unfortunately, the `write` fails, because a local user `joe` is simultaneously writing to every last bit of disk space; `inetd-edit` ends up throwing away all the data in `/etc/inetd.conf`. Write replacement code that preserves the old `/etc/inetd.conf` if there is any problem writing the new data to disk.

Problem 7. The system administrator, after learning that the `/home` disk is full, finds and removes a 40-gigabyte file:

```
% find /home -ls | sort -n +6 | tail -1 | awk '{print $11}'
/home/joe/just-testing/rc
% ls -l /home/joe/just-testing/rc
-rw-r--r-- 1 joe joe 41162685334 Dec 9 10:00 /home/joe/just-testing/rc
% rm /home/joe/just-testing/rc
% ls -l /home/joe/just-testing/rc
ls: /home/joe/just-testing/rc: No such file or directory
%
```

The system administrator later discovers, to his surprise, that the important 16000-byte system file `/etc/rc` has disappeared. What exactly did `joe` do?

Problem 8. The `hapless.edu` HTTP server runs as `root`. Someone connects to the server and asks for the web page `http://hapless.edu/~joe/data/index.html`. In response, the server calls

```
if (!ownedby(fn,user)) return -1;
sendtonetwork(fn);
```

where `fn` is

```
"/home/joe/public_html/data/index.html"
```

and `user` is `"joe"`. The `ownedby()` function checks that the file

```
/home/joe/public_html/data/index.html
```

is owned by `joe`; the `sendtonetwork()` function then reads the file

```
/home/joe/public_html/data/index.html
```

and sends it across the network. Explain how `joe` can read any file on the system.

Problem 9. A setuid-`root` program reads commands from a configuration file set up by the system administrator, and reads arguments from the user running the program. It then writes the commands and arguments to a `root`-owned file:

```
for (i = 0;i < cmdlen;++i) {
  fprintf(fi,"%s ",cmd[i]);
  for (j = 0;arg[i][j];++j) {
    if (!isalpha(arg[i][j])) fprintf(fi,"\\");
    fprintf(fi,"%c",arg[i][j]);
  }
  fprintf(fi," ");
}
```

Another setuid-`root` program reads the file and obeys the instructions:

```
readingcmd = 1; arg_empty(); cmd_empty();
while (fscanf(fi,"%c",&ch) == 1) {
  if (ch == '\\') fscanf(fi,"%c",&ch);
  else if (!isalpha(ch)) {
    if (!readingcmd) { obey(cmd,arg); arg_empty(); cmd_empty(); }
    readingcmd = !readingcmd; continue;
  }
  if (readingcmd) cmd_append(ch); else arg_append(ch);
}
```

Explain how a local user can arrange for `obey("rm","*")` to be called, even if `rm` is not one of the commands specified by the system administrator.

Problem 10. The following program `/bin/su` is installed setuid `root`:

```
#include <pwd.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

FILE *fi;
int uid;
char *args[2];
char line[256];
char user[512];
char pass[512];

int main(int argc,char **argv)
{
  args[0] = getenv("SHELL");
  if (!args[0]) args[0] = "/bin/sh";
  if (!fopen("/dev/null","r")) return 111;
  if (!fopen("/dev/null","r")) return 111;
  if (!fopen("/dev/null","r")) return 111;
  if (!argv[1]) return 111;
  fi = fopen("/etc/actual-passwords","r");
  if (!fi) return 111;
  do
    if (!fgets(line,sizeof line,fi)) return 111;
  while (sscanf(line,"%[^:]:%[^:]:%d",user,pass,&uid) < 3
         || strcmp(user,argv[1]));
  if (!fgets(line,sizeof line,stdin)) return 111;
  if (strlen(line) > 0) line[strlen(line) - 1] = 0;
  if (strcmp(pass,line)) return 111;
  setuid(uid);
  if (getuid() != uid) return 111;
  execv(*args,args);
  return 111;
}
```

The idea is that a user logged in as one account, say `djb`, can run `su faculty`, type the password for the `faculty` account, and have a shell run as `faculty`.

Explain, with complete code, how `joe` can use this `/bin/su` program to print out the contents of `/etc/actual-passwords`. You may assume that `joe`'s password is `494`.