Project 3
ELEC5616 :: Computer and Network Security

Matt Barrie
mattb@ee.usyd.edu.au
Department of Electrical Engineering
University of Sydney

May 20, 2011

Due: To be marked in labs on 3rd June 2011

1 Smashing the Stack

Read "Smashing the Stack For Fun and Profit" by Aleph One.
(See http://www.phrack.com/issues.html?issue=49&id=14#article).

The webserver `commanche` runs as `root` under UNIX and contains the following routine:

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

int printURL(char *myURL) {
    int version = 2;
    char url[256];

    strcpy(url, myURL);
    printf("Loading URL: %s\n", url);

    return 0;
}

void main(int argc, char **argv) {
    printURL(argv[1]);
}
```

`Commanche` is interacted with via the following control code:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <poll.h>
#include <signal.h>

#define DEFAULT_BUFFER_SIZE             512

/* Function to get the address of the start of the stack for the proccess with
 * the given PID
 **/
unsigned long get_sp(pid_t pid) {
    FILE *stat;
    unsigned long sp;
    char filename[1024];
    sprintf(filename, "/proc/%d/stat", pid);
    stat = fopen(filename, "r");
    fscanf(stat, "%*d %*s %*c %*d %*d %*d %*d %*d %*u \
    %*u %*u %*u %*u %*u %*u %*d %*d %*d %*d %*d %*d \
    %*d %*u %*u %*d %*u %*u %*u %lu %*u %*u \
    %*u %*u %*u %*u %*u %*u %*u %*d %*d %*u  \
    %*u", &sp);
    return sp;
}

void main(int argc, char *argv[]) {
    char *buff;     /* The URL string to pass to commanche */
    int bsize=DEFAULT_BUFFER_SIZE;  /* Size for buff */
    int i;
    pid_t pid;      /* Process ID: 0 for child, > 0 for parent */
    int cpipe[2]; /* pipe parent writes to, child reads from */
    int ppipe[2]; /* pipe child writes to, parent reads from */

    if (argc > 1) bsize  = atoi(argv[1]);

    if (!(buff = malloc(bsize))) {
        printf("Can't allocate memory.\n");
        exit(0);
    }
```

```
/* Create two pipes: one for the child to receive data from the parent,
 * one for the parent to receive data from the child */
if(pipe(cpipe)) {
    fprintf(stderr, "Pipe failed\n");
    exit(-1);
}
if(pipe(ppipe)) {
    fprintf(stderr, "Pipe failed\n");
    exit(-1);
}

/* Create a child process */
if((pid = fork()) == -1) {
    perror("fork");
    exit(-1);
}
if(pid == (pid_t)0) {
    /* Child process */

    /* Set up pipes */
    close(cpipe[1]);
    dup2(cpipe[0],0);
    close(cpipe[0]);

    close(ppipe[0]);
    dup2(ppipe[1],1);
    close(ppipe[1]);

    /* Run commanche */
    execlp("./commanche", "./commanche", NULL);

    /* Check for errors */
    perror("execlp");
    _exit(-1);
}
else {
    /* Parent process */
    FILE *outstream;
    char line[2048];
    int status;
    struct pollfd fds[2];

    /* Close the ends of the pipes the parent doesn't use */
```

```
    close(cpipe[0]);
    close(ppipe[1]);

    outstream = fdopen(cpipe[1], "w");

    /* Write the URL string to the commanche process */
    for(i = 0; i < bsize; i++)
        fputc(buff[i], outstream);
    fputc('\n', outstream);
    fflush(outstream);

    /* Set up file descriptor structs for poll() */
    fds[0].fd = 0;
    fds[0].events = POLLIN | POLLPRI;
    fds[1].fd = ppipe[0];
    fds[1].events = POLLIN | POLLPRI;

/* Read loop: Reads data from child and echoes to stdout, reads data
 * from stdin and echoes to child. This allows us to interact with the
 * commanche process from the parent */
while(1) {
    int res;
    if((res = poll(fds, 2, -1)) > 0) {
        int bytes;
        if(fds[0].revents & POLLIN || fds[0].revents & POLLPRI) {
            if((bytes = read(0, line, 2046)) > 0) {
                line[bytes + 1] = '\0';
                fprintf(outstream, "%s\n", line);
                fflush(outstream);
            }
            else if(bytes < 0) {
                perror("read from stdin");
                break;
            }
            else {
                printf("EOF on stdin\n");
                break;
            }
        }
        if(fds[1].revents & POLLIN || fds[1].revents & POLLPRI) {
            if((bytes = read(ppipe[0], line, 2046)) > 0) {
                line[bytes + 1] = '\0';
                printf("%s\n", line);
                fflush(stdout);
```

```
                }
                else if(bytes < 0) {
                    perror("read from child");
                    break;
                }
                else {
                    printf("EOF from child\n");
                    break;
                }
            }
        }
        else if (res < 0) {
            perror("poll");
            break;
        }
    }

    /* Wait for the child to exit and check whether it was terminated by a signal */
    wait(&status);
    if(WIFSIGNALED(status))
        printf("Child terminated by signal %d\n", WTERMSIG(status));
    else
        printf("Child exited normally\n");
    }
}
```

2 Questions

A. What problem exists in the above code?

B. Write a program to exploit this problem and obtain a root shell. Fully comment your code and demonstrate to the tutors that it works.

C. How would you rewrite the routine to avoid this problem?

D. If commanche was written in Java would this problem still exist?

E. Can you think of any variations on the exploit?

F. As far as Internet security goes, do you think this problem is more or less important than the lack of strong authentication in the IP stack?

Answer the questions above in no more than four pages. Ensure your code is legible and well documented.