Some simple experiments with buffer overflows and canaries

Hayo Thielecke
http://www.cs.bham.ac.uk/~hxt
Vulnerable code

```c
f()
{
    char buf[4];

    gets(buf);
}
```
Try without canaries

wallace% gcc buftest.c -o buftest
/tmp/ccMlQXTU.o: In function ‘f’: 
buftest.c:(.text+0x2c): warning: the ‘gets’ function is dangerous and should not be used.
wallace% buftest
It was a dark and stormy night
Segmentation fault
Try with canaries

wallace% gcc buftest.c -o buftest -fstack-protector-all
/tmp/cc0fKBlY.o: In function ‘f’:
buftest.c: (.text+0x53): warning: the ‘gets’ function is dangerous

wallace% buftest
It was a dark and stormy night
*** stack smashing detected ***: buftest terminated
Compiled without canaries

```assembly
f:
pushl %ebp
movl %esp, %ebp
subl $24, %esp
leal -4(%ebp), %eax
movl %eax, (%esp)
call gets
leave
ret
```
Function prologue and epilogue

leave means
leave stack frame, equivalent to

```
mov %ebp, %esp
pop %ebp
```

leal = Load Effective Address
f:
pushl %ebp
movl %esp, %ebp
subl $24, %esp
movl %gs:20, %eax
movl %eax, -4(%ebp)
xorl %eax, %eax
leal -8(%ebp), %eax
movl %eax, (%esp)
call gets
movl -4(%ebp), %edx
xorl %gs:20, %edx
je .L6
call __stack_chk_fail
.L6:
leave
ret
Now we attack local variables, not the return address

```c
f()
{
    char  buf1[4];
    char  buf2[4];

    gets(buf1);
    gets(buf2);

    printf("%s\n", buf1);
    printf("%s\n", buf2);
}
```
We can write into another variable, even with canaries

```plaintext
wallace% gcc buftest2.c -o buftest2 -fstack-protector-all
/tmp/cceKZ0no.o: In function ‘f’:
buftest2.c:(.text+0x53): warning: the ‘gets’ function is dangerous and should not be used.
wallace% buftest2
aa
bbbbbbb
bbb
bbbbbbb
```
Now we allocate the strings in the heap, not the stack

```c
f()
{
    char *buf1 = (char *)malloc(4);
    char *buf2 = (char *)malloc(4);

    gets(buf1);
    gets(buf2);

    printf("buf1 = \\
    \\
", buf1);
    printf("buf2 = \\
    \\
", buf2);

    printf("We deallocate the strings in the heap... \\
    \\
");
    free(buf1);
    free(buf2);
}
```
Strings can be overflowed, crash on deallocating, not canary

```
wallace% buftest-heap
01234567890123456789
bbb
buf1 =
0123456789012345bbb

buf2 =
bbb

We deallocate the strings in the heap...

*** glibc detected *** buftest-heap: free(): invalid next size (fast): 0x09bb4008 ***
======== Backtrace: ========
...
```
int n;
char c[] = "Smith";
int a = 1234;
int i;

printf("Dear Mr %s, please pay $%d immediately.%n", c, a, &n);
putchar(’\n’);
for(i = 0; i < n; i++) putchar(’-’);
putchar(’\n’);

Output:

Dear Mr Smith, please pay $1234 immediately.
--------------------------------------------
Format string attack - reading from the stack frame

f()
{
    int x = 111111111;
    int y = 222222222;
    int z = 333333333;

    char *format;

    format = (char *)malloc(80);

    gets(format);

    printf(format);

}
Format string attack - reading from the stack frame

f:
pushl %ebp
movl %esp, %ebp
subl $24, %esp
movl $111111111, -16(%ebp)
movl $222222222, -12(%ebp)
movl $333333333, -4(%ebp)
movl $80, (%esp)
call malloc
movl %eax, -8(%ebp)
movl -8(%ebp), %eax
movl %eax, (%esp)
call gets
movl -8(%ebp), %eax
movl %eax, (%esp)
call printf
leave
ret
Format string attack - reading from the stack frame

wallace% format-test
%d, %d, %d, %d, %d
4330012, 111111111, 222222222, 153014280, 333333333

wallace% format-test
%d,%d,%d,%s and the format string again
4330012,111111111,222222222,%d,%d,%d,%s
and the format string again
and the format string again

If we had used canaries, the format string attack could also see those.
Use-after-free vulnerability

f()
{
    char *q;
    char *p = (char *)malloc(80);

    strcpy(p, "This text is not user-malleable.\n");
    printf("p points to \n%s\n", p);
    free(p);

    q = (char *)malloc(80);
    gets(q);

    printf("p now points to \n%s\n", p);
}
Use-after-free vulnerability

wallace% use-after-free
p points to
This text is not user-malleable.

I IZ IN UR HEAP
p now points to
I IZ IN UR HEAP