CS 577 Cybersecurity Lab

Lab 5 – due 10/22/14 11:59pm

Subject: Buffer Overflow Protection Using Guard Pages

Aim of this deliverable is to protect dynamically allocated buffers from overflows by providing **guard pages** (see figure 1). Your task is to develop your own version of **5** popular functions responsible for memory allocation in the C program language. You need to implement versions that protect from read & write overflows

HINT: The mprotect() system call can modify the protection of memory pages (e.g., to make then non-readable & writable). The mmap() system call can map (i.e., request) memory from the operating system. These calls operate on memory pages.

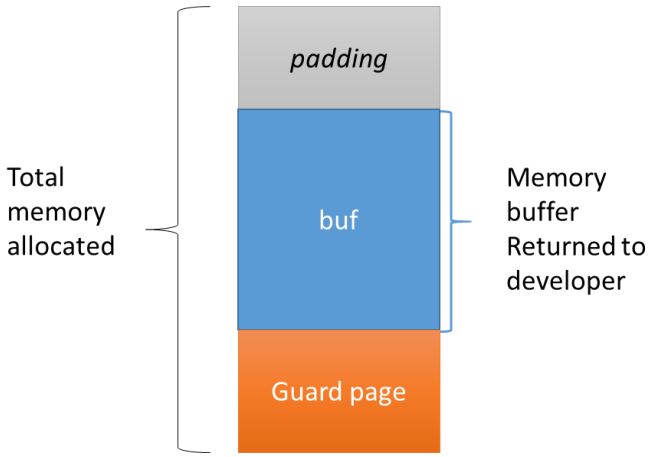

Figure 1. A buffer protected by a guard page

Deliverables

A shared library that implements the following functions. [80%]

1. **pmalloc() [20%]**
   Create pmalloc() that overrides malloc() and provides guard-page protection.
   malloc() -Returns a pointer to a block of at least size bytes. Notice that some programs expect that the memory returned by malloc() is 4 or 8 byte aligned

(4→32-bit, 8→64bit). If the space assigned by malloc() is overrun, the results are undefined. Use *man malloc* for more information.
HINT: If your pmalloc() returns properly aligned memory, you may need to also include padding before the guard page.

2. **pfree() [20%]**
   Create pfree() that overrides free() and supports buffers allocated with pmalloc().
   free()- De-allocates a pointer to a block previously allocated by malloc(), calloc(), or realloc(). After free() is executed, this space is made available for further allocation by the application, though not returned to the system. Memory is returned to the system only upon termination of the application. If ptr is a null pointer, no action occurs. If a random number is passed to free(), the results are undefined. Use *man free* for more information.
3. **pcalloc() [10%]**
   Create pcalloc() that overrides calloc() and provides guard-page protection.
   calloc() - Allocates a block of memory for an array of num elements, each of them size bytes long, and initializes all its bits to zero. Use *man calloc* for more information.
4. **prealloc() [20%]**
   Create prealloc() that overrides realloc() and provides guard-page protection.
   realloc() - Changes the size of the block pointed to by ptr to size bytes and returns a pointer to the (possibly moved) block. Use *man realloc* for more information.
5. **pmemalign() [10%] – challenging**
   Create pmemalign () that overrides memalign() and provides Guard page protection.
   memalign() - Allocates size bytes on a specified alignment boundary and returns a pointer to the allocated block. The value of the returned address is guaranteed to be an even multiple of alignment. The value of alignment must be a power of two and must be greater than or equal to the size of a word. Use *man memalign* for more information.

Include a report.txt file explaining how your function wrappers work and what kind of attacks they prevent. Do they protect the programs you exploited in the previous two labs? [20%]

Submission information

The code you submit for grading must build and run on the linux-lab, even if you use a different machine/environment for developing. You should use a particular host in the linux-lab for this assignment, which we have tested and will more likely result into consistently working exploits.

Submit all your files as a tar.gz archive through Canvas.