CS 577 Cybersecurity Lab

Stevens Institute of Technology

Lab 4 – due 10/09/14 6:15pm

Instructor Georgios Portokalidis

Teaching Assistant Dimitrios Damopoulos

This assignment will help you learn how linking and loading is performed by the OS after a program is compiled, how function interposition works, and how buffers can be protected by overriding the allocation functions with new ones that support guard pages.

This lab will be done on the linux-lab.

You will need ssh, PuTTY, and a terminal to connect.

Protect Memory Allocation with Guard Pages

Aim of this deliverable is to protect memory allocation by providing **Guard pages** and protect the buffers (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, while as a bonus you can create a mixed version that protects both from over and underflows (reading or writing before a pointer).

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.

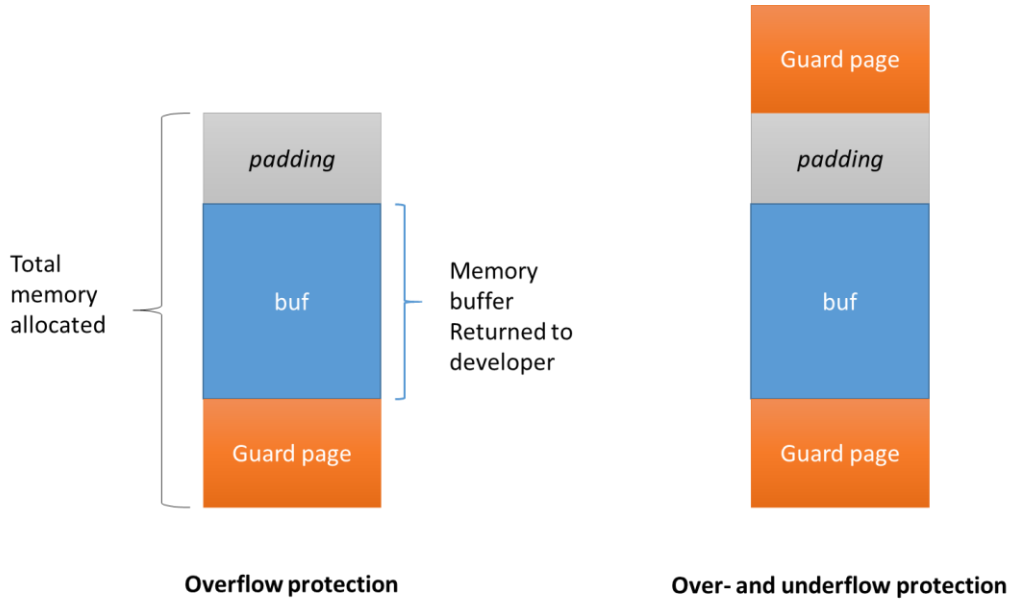

Figure 1. Guard Pages

Exercise 1. pmalloc() (30%)

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.

Link: http://www.cplusplus.com/reference/cstdlib/malloc/

HINT: If your pmalloc() returns properly aligned memory, you may need to also include padding before the guard page.

Exercise 2. pfree() (20%)

Create pfree() that overrides free() and support Guard page protection.

free()- Is 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.

Link: http://www.cplusplus.com/reference/cstdlib/free/

Exercise 3. calloc() (15%)

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.

Link: http://www.cplusplus.com/reference/cstdlib/calloc/

Exercise 4. realloc() (15%)

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.

Link: http://www.cplusplus.com/reference/cstdlib/realloc/

Exercise 5. Testing & Reporting (20%)

You have to create simple test programs that utilize the aforementioned functions. Provide a report on how the programs behave with and without the Guard pages when a buffer overflow occurs.

Bonus 1. pmemalign() (10%)

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.

Link: http://linux.die.net/man/3/memalign

Hint: You may need to include padding between the buffer and the guard page as well.

Bonus 2. Mixed Guard pages (10%)

A Mixed version of Guard pages provides both Overwrite and Underwrite protection.

Grading

As always, we will grade your work on quality from both the user's and programmer's points of view. Each program should contain function-level and local comments as appropriate, as well as an explanation of the program's principles of operation. **PLEASE SUBMIT**: ***Documentation***, ***your source code with comments***, ***a report with all the steps and problems you had***.