Agenda

- Heap Review
  - Memory location of Heap and BSS
- Heap and BSS
  - Non-executable
  - executable
- Vulnerable code
- Verify exploitation
- Sensitive heap data of functions
- Reference
# Memory location of Heap and BSS

<table>
<thead>
<tr>
<th>Segment</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>text (code) segment</td>
<td>Low address</td>
</tr>
<tr>
<td>data segment</td>
<td></td>
</tr>
<tr>
<td>bss segment</td>
<td></td>
</tr>
<tr>
<td>heap segment</td>
<td>Low address</td>
</tr>
<tr>
<td>stack segment</td>
<td>High address</td>
</tr>
</tbody>
</table>

- The heap grows down toward higher memory addresses.
- The stack grows up toward lower memory addresses.
Heap and BSS

- Non-executable
  - Heap
    - continued declaration of variable
      ```c
      char *userinput = malloc(20);
      char *outputfile = malloc(20);
      ```
  - BSS
    - Just like Heap, but static declaration
      ```c
      static char userinput[BUFSIZE];
      static char outputfile[BUFSIZE];
      ```
Heap and BSS (cont.)

- **Executable**
  - Exploiting function pointers
  - Allows to dynamically modify a function
    
    ```
    ex: inf (*funcptr)(char *str);
    ```
Vulnerable code

```c
int goodfunc(const char *str); /* funcptr start out as this */
int main(int argc, char **argv)
{
    static char buf[BUFSIZE];
    static int (*funcptr)(const char *str);
    funcptr = (int (*)(const char *str))goodfunc;
    memset(buf, 0, sizeof(buf));
    strncpy(buf, argv[1], strlen(argv[1]));
    /* This is what funcptr would point to if we didn’t overflow it */
    int goodfunc(const char *str)
    {
        blahblah;
    }
}
```

2005-02-19
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define ERROR -1
#define BUFSIZE 64

int goodfunc(const char *str); /* funcptr starts out as this */

int main(int argc, char **argv)
{
    static char buf[BUFSIZE];
    static int (*funcptr)(const char *str);
    
    if (argc <= 2)
    {
        fprintf(stderr, "Usage: %s <buf> <goodfunc arg>
        exit(ERROR);
    }

    funcptr = (int (*)(const char *str))goodfunc;
    printf("before overflow: funcptr points to %p\n", funcptr);
    memset(buf, 0, sizeof(buf));
    strncpy(buf, argv[1], strlen(argv[1]));
    printf("after overflow: funcptr points to %p\n", funcptr);
    (void)(*funcptr)(argv[2]);
    return 0;
}

int goodfunc(const char *str)
{
    printf("\nHi, I'm a good function. I was passed: %s\n", str);
    return 0;
}
```
Hi. I'm a good function. I was passed: kids before overflow: funcptr points to 0x804869c after overflow: funcptr points to 0x8048631 illegal instruction before overflow: funcptr points to 0x804869c after overflow: funcptr points to 0x8048634 illegal instruction before overflow: funcptr points to 0x804869c after overflow: funcptr points to 0x8048635 Bus error before overflow: funcptr points to 0x804869c after overflow: funcptr points to 0x8048637 Segmentation fault before overflow: funcptr points to 0x804869c after overflow: funcptr points to 0x8048638 Segmentation fault
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630
after overflow: funcptr points to 0x8048630

[cups|eintsys|W1] ~/heap >
Feb/19 Sat 05:08 AM 0 csh [1 csh]
Exploiting method

- system() method
  - easily to guess the address of system
  - change to `system("/bin/sh");`
  - fairly quickly

- argv[] method
  - store the shellcode in an argument to the program (requiring an executable heap)
  - don’t require compatible function pointers
    - `char (*funcptr)(int a); = void (*funcptr)();`

- Heap offset method
  - offset from the top of the heap to the estimated address of the target/overflow buffer (requiring an executable heap)
## Sensitive heap data of functions

(from w00w00)

<table>
<thead>
<tr>
<th>Functions</th>
<th>Examples include</th>
</tr>
</thead>
<tbody>
<tr>
<td>*gets()/*printf(), *scanf()</td>
<td>_iob (FILE) structure in heap</td>
</tr>
<tr>
<td>popen()</td>
<td>_iob (FILE) structure in heap</td>
</tr>
<tr>
<td>*dir() (readdir, seekdir,...)</td>
<td>DIR entries (dir/heap buffers)</td>
</tr>
<tr>
<td>atexit()</td>
<td>static/global function pointers</td>
</tr>
<tr>
<td>strdup()</td>
<td>Allocates dynamic data in the heap</td>
</tr>
<tr>
<td>getenv()</td>
<td>Stored data on heap</td>
</tr>
</tbody>
</table>
# Sensitive heap data of functions

(from w00w00)

<table>
<thead>
<tr>
<th>Functions</th>
<th>Examples include</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmpnam()</td>
<td>Stored data on heap</td>
</tr>
<tr>
<td>Malloc()</td>
<td>Chain pointers</td>
</tr>
<tr>
<td>rpc callback function</td>
<td>Function pointers</td>
</tr>
<tr>
<td>windows callback functions</td>
<td>Func pointers kept on heap</td>
</tr>
<tr>
<td>signal handler pointer in cygnus (gcc for win)</td>
<td>Functions pointers (note: unix tracks theses in the kernel, not in the heap)</td>
</tr>
</tbody>
</table>
Reference

- http://www.w00w00.org/files/heaptut/
  - Chinese version
  - English version

- Hacking – The Art of Exploitation
  - By Jon Erickson
  - ISBN 1-59327-007-0