Buffer overflow

Mohammad Mahzoun

University of Tehran

May 24, 2018

Introduction

What is a buffer overflow?

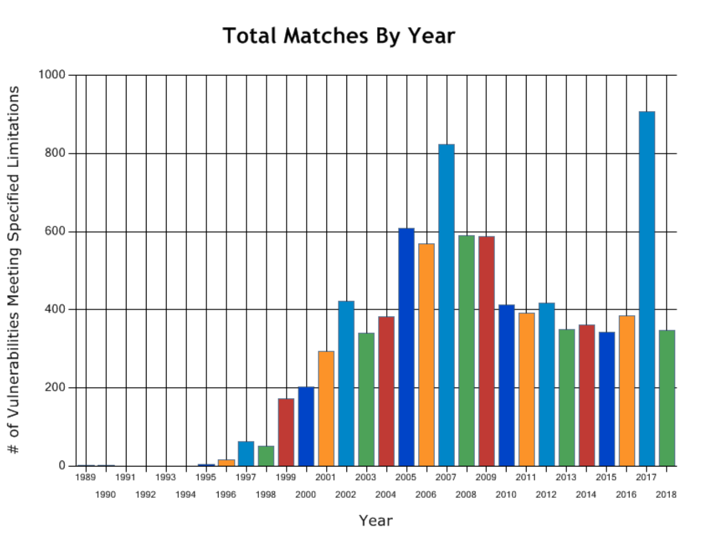
- A buffer overflow is a bug that affects low-level code, typically written in C and C++, with significant security implications.
- A program with this bug will simply crash.
- But an Attacker can do much worse!
  - **Steal** private information.
  - **Corrupt** valuable information.
  - **Run** arbitrary code.

History

History of buffer overflows

- Morris worm (1988)
  - Propagated across the machines using buffer overflow.
  - End result: $10-100M in damages
- CodeRed (2001)
  - Exploited an overflow in MS-IIS server
  - 300,000 machines infected in 14 hours
- X11 Vulnerability (2014)
  - The bug was in code for more than 20 years.

History



https://nvd.nist.gov/vuln/search/statistics

C memory layout

A typical memory representation of C program consists of following sections:

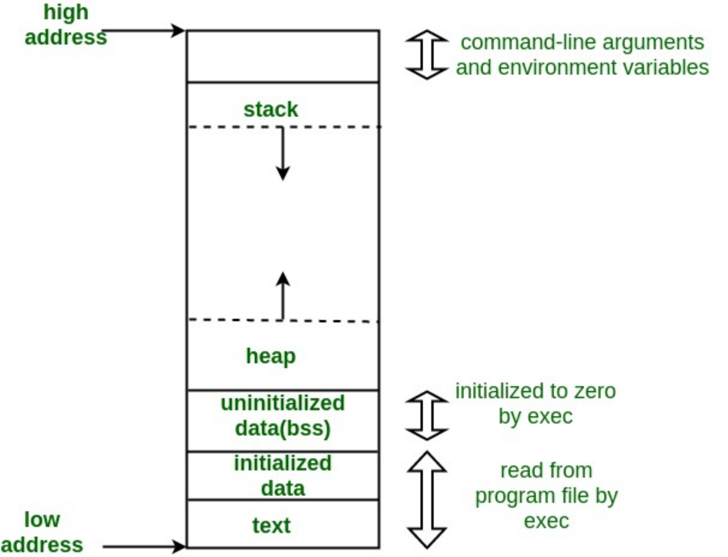
- Text segment
  - contains executable instructions.
  - Placed below the heap or stack in order to prevent heaps and stack overflows from overwriting it.
- Initialized data segment
  - virtual address space contains the global variables and static variables initialized.
- Uninitialized Data Segment
  - bss (block started by symbol)
  - all global variables and static variables that are initialized to zero or do not have explicit initialization

C memory layout

A typical memory representation of C program consists of following sections:

- Stack
  - local variables variables
  - saved information after function calls
- Heap
  - begins at the end of the BSS segment and grows to larger addresses from there.
  - managed by malloc, realloc, and free, which may use the brk and sbrk system calls to adjust its size.

C memory layout

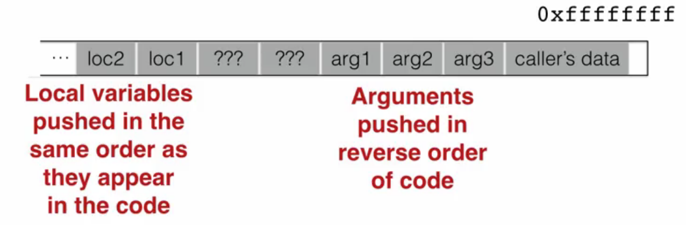

high address
command-line arguments and environment variables
stack
heap
uninitialized data(bss)
initialized to zero by exec
initialized data
text
read from program file by exec
low address

Stack and function calls

- what happens when we call a function?
  - what data needs to be stored?
  - where does it go?
- what happens when we return from a function?
  - what data needs to be restored?
  - where does come from?

Stack and function calls

```
void func(char* arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    ...
}
```

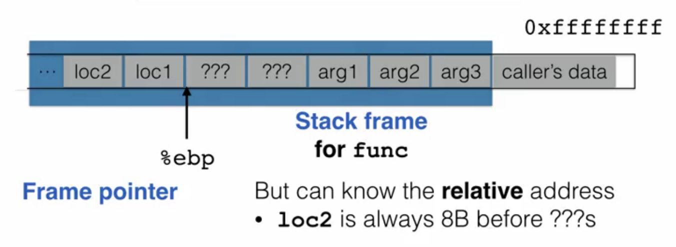


Local variables pushed in the same order as they appear in the code

Arguments pushed in reverse order of code

Accessing variables

```
void func(char* arg1, int arg2, int arg3)
{
    ...
    loc2++; // Where is it? %ebp - 8
    ...
}
```

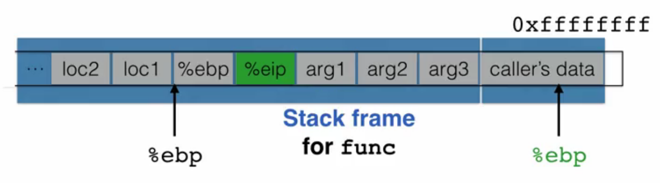


But can know the **relative** address

- **loc2** is always 8B before ???s

Returning from a function

```
int main()
{
    ...
    func("Hey", 10, -3);
    ...
}
```

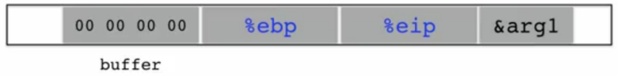


Buffer overflows

- Buffer
  - Contiguous memory associated with a variable or field
  - Common in C (Strings)
- Overflow
  - Put more into a buffer than it can hold
- Where does the overflowing data go?
- Well, now we know the memory layout ...

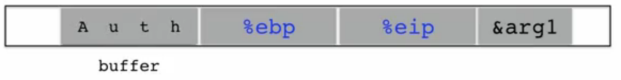
Buffer overflows

```
void func(char *arg1)
{
   char buffer[4];
   strcpy(buffer, arg1);
    ...
}
int main()
{
     const *mystr = "Authme!";
     func(mystr);
      ...
}
```



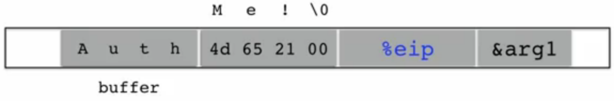
Buffer overflows

```
void func(char *arg1)
{
   char buffer[4];
   strcpy(buffer, arg1);
    ...
}
int main()
{
    const *mystr = "Authme!";
    func(mystr);
     ...
}
```



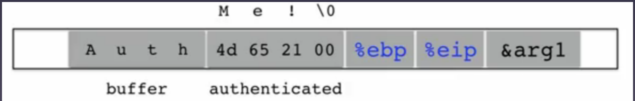
Buffer overflows

```
void func(char *arg1)
{
   char buffer[4];
   strcpy(buffer, arg1);
    ...
}
int main()
{
     const *mystr = "Authme!";
     func(mystr);
      ...
}
```



Buffer overflows

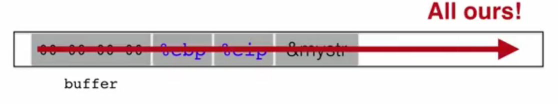
```
void func(char *arg1)
{
   int authenticated = 0
   char buffer[4];
   strcpy(buffer, arg1);
   if(authenticated) { ...
}
int main()
{
    const *mystr = "Authme!";
    func(mystr);
     ...
}
```



```
                 M   e   !   \0
|   | A  u  t  h | 4d 65 21 00 | %ebp | %eip | &arg1 |   |
         buffer    authenticated
```

Even Worse!

Attacker can inject his code and arrange for the program to execute it!



Questions