Vulnerability Discovery & Exploitation

Week 2 Integer Overflow

Outline

- Integer
- Integer overflow/underflow
- Command-line arguments
- Variable Scoping
- Memory Segmentation

Lab sessions

- Lab sessions are delivered via face-to-face in LG007.
- Please note
  - They are not drop in sessions
  - Attend the lab session according to your timetable
  - The chance of someone failing the module is high if they do not attend the lab session
- C programming tutorial for beginners https://www.youtube.com/watch?v=KJgsSFOSQv0
- Reading: Hacking: The art of exploitation, 2nd Edition chapters 2-3 and 5-6

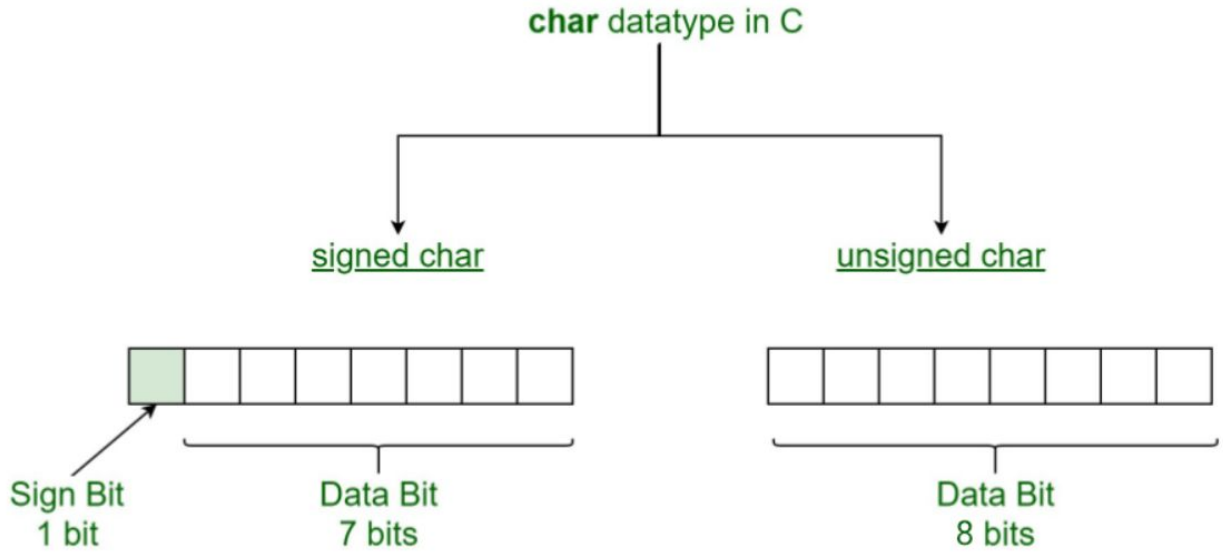
Integer

- A positive or negative whole number with no fractional components; it has a fixed boundary.

Type	Storage size	Value range
char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
int	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned int	4 bytes	0 to 4,294,967,295
long	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Integer

- A positive or negative whole number with no fractional components; it has a fixed boundary.



Integer overflow

- Also known as wraparound, occurs when an arithmetic operation outputs a numeric value that falls outside allocated memory space or overflows the range of the given value of the integer.

Integer overflow

- demo

What happens with Integer Overflow

- `unsigned char i=255;`
- `i=i+1;`
- `What is the value of i?`

```
255 in binary     11111111

1 in binary       00000001

255+1 in binary  100000000
```

What happens with Integer underflow

- unsigned char i=0;
- i=i-1;
- What is the value of i?

```
0 in binary          00000000

-1 in binary         11111111

0+(-1) in binary     11111111=255
```

Integer overflow attacks

cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html

Introduction

The 2021 Common Weakness Enumeration (CWE™) Top 25 Most Dangerous Software Weaknesses (CWE Top 25) is a demonstrative list of the most common and impactful issues experienced over the previous two calendar years. These weaknesses are dangerous because they are often easy to find, exploit, and can allow adversaries to completely take over a system, steal data, or prevent an application from working. The CWE Top 25 is a valuable community resource that can help developers, testers, and users — as well as project managers, security researchers, and educators — provide insight into the most severe and current security weaknesses.

To create the 2021 list, the CWE Team leveraged Common Vulnerabilities and Exposures (CVE®) data found within the National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD), as well as the Common Vulnerability Scoring System (CVSS) scores associated with each CVE record. A formula was applied to the data to score each weakness based on prevalence and severity.

The CWE Top 25

Below is a brief listing of the weaknesses in the 2021 CWE Top 25, including the overall score of each.

Rank	ID	Name	Score	2020 Rank Change
[1]	CWE-787	Out-of-bounds Write	65.93	+1
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.84	-1
[3]	CWE-125	Out-of-bounds Read	24.9	+1
[4]	CWE-20	Improper Input Validation	20.47	-1
[5]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	19.55	+5
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	19.54	0
[7]	CWE-416	Use After Free	16.83	+1
[8]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.69	+4
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	14.46	0
[10]	CWE-434	Unrestricted Upload of File with Dangerous Type	8.45	+5
[11]	CWE-306	Missing Authentication for Critical Function	7.93	+13
[12]	CWE-190	Integer Overflow or Wraparound	7.12	-1

Command Line arguments

- In C, command-line arguments can be accessed in the main()function by including two additional arguments to the function: an integer and a pointer to an array of strings.

```c
#include<stdio.h>
int main(int argc, char **argv){
        int i;
        printf("There were %d arguments provided:\n", argc);
        for(i=0;i<argc;i++){
           printf("argument #%d is %s\n", i, argv[i]);
        }
}
```

Command Line arguments

- In C, command-line arguments can be accessed in the main()function by including two additional arguments to the function: an integer and a pointer to an array of strings.

```
fly@VM:~/vde/week2$ ./command_line 1 2 3 VDE
There were 5 arguments provided:
argument #0 is ./command_line
argument #1 is 1
argument #2 is 2
argument #3 is 3
argument #4 is VDE
```

Variable scoping

- Local variables: are inside a function or a block

```
#include<stdio.h>
int main(){

/* local variables declaration */
int a, b, c;
/* actual initialization */
a = 1;
b = 2;
c = a+b;
printf(“value of a = %d, b = %d, and c = %d”, a, b, c);

}
```

Variable scoping

- Global variables: are outside of all functions

```
#include<stdio.h>
/* global variable declaration */
int g;
int main(){

/* local variables declaration */
int a, b;
/* actual initialization */
a = 1;
b = 2;
g = a+b;
printf(“value of a = %d, b = %d, and g = %d”, a, b, g);

}
```

Variable scoping

- Global variables: are outside of all functions

```
#include<stdio.h>
/* global variable declaration */
int g=20;
int main(){

/* local variables declaration */
int g =10;
printf(“value of g = %d”, g);

}
```

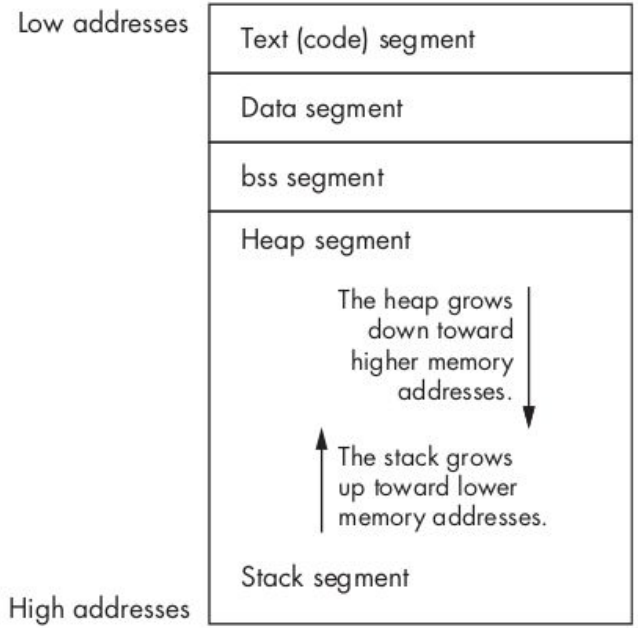
Variable scoping

- Static variables: remains intact between function calls; they are initialized once.

```
#include<stdio.h>

void function(){
        int var = 5;
        static int static_var = 5;

        printf("\t[in function] var = %d\n", var);
        printf("\t[in function] static var = %d\n", static_var);
        var++;
        static_var++;
}
int main(){
        int i;
        static int static_var = 1337;

        for(i=0;i<5;i++){
            printf("[in main] static_var = %d\n", static_var);
            function();
        }
}
```

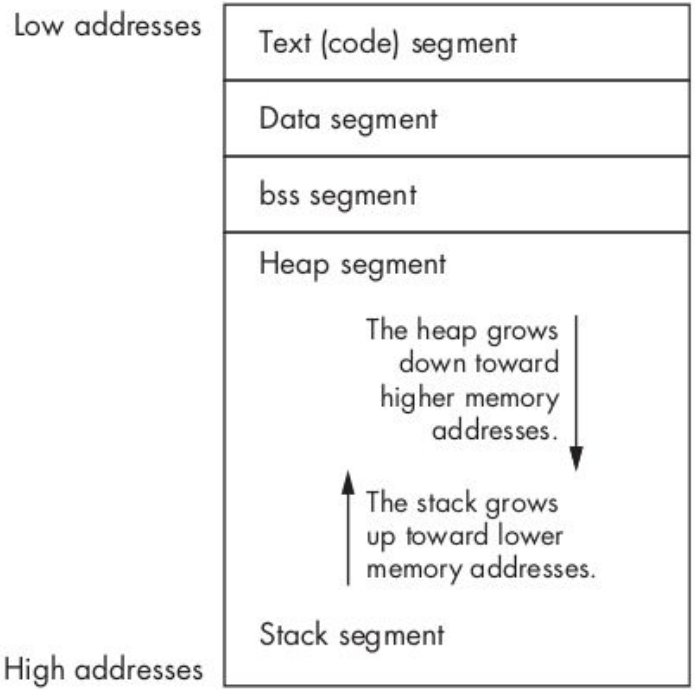
Memory Segmentation

- Text segment is where the assembled machine language instructions of the program are located.
- Write permission is disabled in the text segment.
- It has a fixed size.



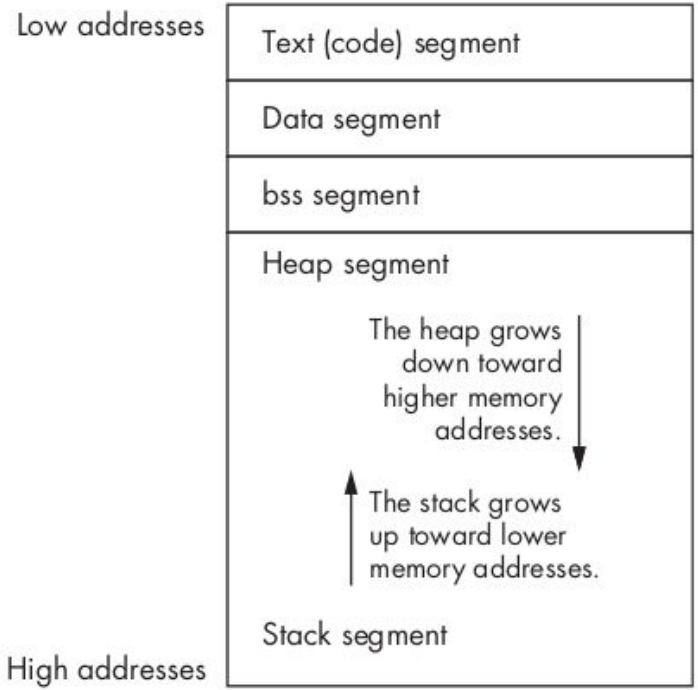
Memory Segmentation

- Data segment is filled with the initialised global and static variables
- bss segment is filled with uninitialised global and static variables
- Both data and bss segments also have a fixed size



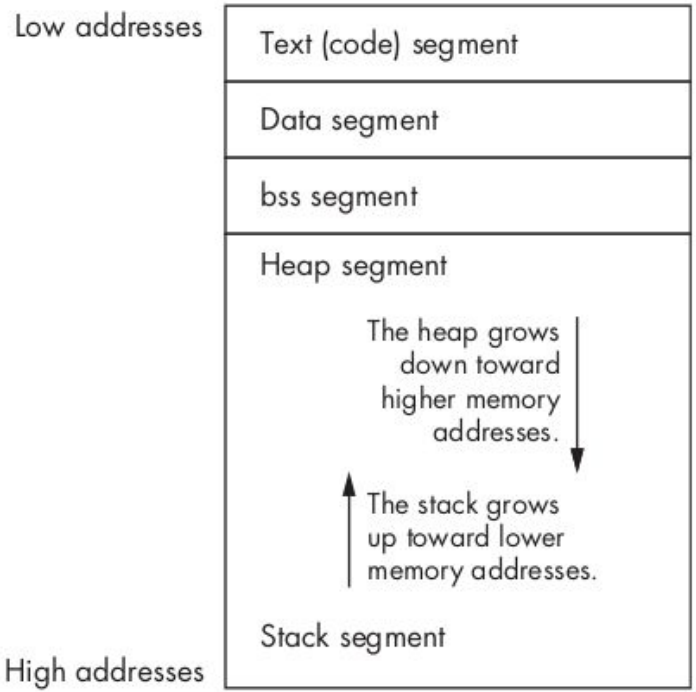
Memory Segmentation

- Heap segment is a segment of memory a programmer can directly control.
- Memory within the heap is managed by allocator and deallocator algorithms (e.g. malloc, free)
- The growth of the heap moves downward toward higher memory address.



Memory Segmentation

- Stack segment also has variable size and is used as a temporary scratch pad to store local function variables and context during functions calls.
- The growth of the stack moves upward toward low memory address.



Memory Segmentation

- demo