X

Week 4 : Assignment 4

The due date for submitting this assignment has passed.

Due on 2025-02-19, 23:59 IST.

Assignment submitted on 2025-02-18, 21:27 IST

1) Which of the following code snippets is most likely to cause an integer overflow vulnerability? ***1 point***

- int a = INT_MAX;
  int b = 1;
  int result = a + b;
- int a = 100;
  int b = 200;
  int result = a - b;
- int a = 10;
  int b = 20;
  int result = a * b;
- int a = 1000;
  int b = 10;
  int result = a / b;

Yes, the answer is correct.
Score: 1

Accepted Answers:

int a = INT_MAX;
int b = 1;
int result = a + b;

2) What is the primary danger of the following heap overflow code? ***1 point***

```
char *buffer = (char *)malloc(10);
strcpy(buffer, "This is a very long string that exceeds the buffer size");
```

- (●) It can corrupt adjacent memory chunks, leading to arbitrary code execution.
- ( ) It can cause the program to crash due to a segmentation fault.
- ( ) It can overwrite the return address on the stack.
- ( ) It can cause a denial of service by exhausting memory.

Yes, the answer is correct.
Score: 1

Accepted Answers:
It can corrupt adjacent memory chunks, leading to arbitrary code execution.

3) In glibc's heap implementation, what is the purpose of the "top chunk" in the following code? ***1 point***

```
void *chunk1 = malloc(100);
void *chunk2 = malloc(200);
free(chunk1);
free(chunk2);
```

- ( ) It is the first chunk allocated in the heap.
- (●) It is a special chunk that serves as the boundary between allocated and free memory.
- ( ) It is used to store metadata about the heap.
- ( ) It is a chunk that is always free and never allocated.

Yes, the answer is correct.
Score: 1

Accepted Answers:
It is a special chunk that serves as the boundary between allocated and free memory.

4) Which of the following code snippets demonstrates a common technique to exploit heap metadata corruption? ***1 point***

- ( )
```
char *buffer = (char *)malloc(10);
free(buffer);
buffer[0] = 'A';
```
- ( )
```
char *buffer = (char *)malloc(10);
buffer[10] = 'A';
```
- (●)
```
char *buffer = (char *)malloc(10);
free(buffer);
*(size_t *)(buffer - 8) = 0x41;
```
- ( )
```
char *buffer = (char *)malloc(10);
strcpy(buffer, "AAAAAAAAAA");
```

Yes, the answer is correct.

unit=43&lesson=51)

Quiz: Week 4 : Assignment 4 (assessment?name=148)

Week 5 ()

Week 6 ()

Week 7 ()

Week 8 ()

Download Videos ()

Text Transcripts ()

Books ()

Lecture Material ()

Score: 1

Accepted Answers:

*char *buffer = (char *)malloc(10);*

free(buffer);

**(size_t *)(buffer - 8) = 0x41;*

5) What happens if an integer overflow occurs in the size calculation for a memory allocation request in the following code? ***1 point***

```
size_t size = 100;
size_t count = SIZE_MAX / size + 1;
void *buffer = malloc(size * count);
```

- The program will crash immediately.
- A smaller buffer than expected may be allocated, leading to a buffer overflow.
- The heap will be corrupted, and all future allocations will fail.
- The program will enter an infinite loop.

Yes, the answer is correct.
Score: 1

Accepted Answers:

A smaller buffer than expected may be allocated, leading to a buffer overflow.

6) In glibc's heap implementation, what is the purpose of heap consolidation? ***1 point***

- To merge adjacent free chunks to reduce fragmentation.
- To allocate memory more efficiently.
- To prevent double-free vulnerabilities.
- To detect heap overflows.

Yes, the answer is correct.
Score: 1

Accepted Answers:

To merge adjacent free chunks to reduce fragmentation.

7) Consider the following code snippet: ***1 point***

```
#include <stdlib.h>
#include <string.h>

int main() {
    char *buffer1 = (char *)malloc(16);
    char *buffer2 = (char *)malloc(16);
    free(buffer1);
    free(buffer2);
    char *buffer3 = (char *)malloc(16);
    strcpy(buffer3, "Exploit Me!");
    strcpy(buffer1, "Overwrite!");

    return 0;
```

```
}
```

- The program will crash due to a segmentation fault
- The contents of `buffer3` will be overwritten with "Overwrite!"
- The heap metadata will be corrupted, leading to arbitrary code execution
- The program will terminate normally without any visible effects

Yes, the answer is correct.
Score: 1

Accepted Answers:
The contents of `buffer3` will be overwritten with "Overwrite!"

8) In glibc's heap implementation, which of the following best explains why `buffer3` reuses the memory previously occupied by `buffer1`? ***1 point***

- The heap allocator uses a first-fit strategy to allocate memory.
- The heap allocator uses a last-in-first-out (LIFO) strategy for fastbins.
- The heap allocator merges adjacent free chunks to reduce fragmentation.
- The heap allocator uses a best-fit strategy to allocate memory.

Yes, the answer is correct.
Score: 1

Accepted Answers:
The heap allocator uses a last-in-first-out (LIFO) strategy for fastbins.

9) In glibc's heap implementation, the `prev_size` field of a chunk is only used when the previous chunk is free. ***1 point***

- True
- False

Yes, the answer is correct.
Score: 1

Accepted Answers:
True

10) In glibc's heap implementation, the size of a chunk is always aligned to 8 bytes (on 32-bit systems) or 16 bytes (on 64-bit systems). If a user requests 24 bytes of memory on a 64-bit system, what will be the actual size of the allocated chunk (including metadata)?

48

No, the answer is incorrect.
Score: 0

Accepted Answers:
(Type: Numeric) 32

1 point