Week 1 quiz

15 questions

1
point

1.
Three of the following are classic security properties; which one is not?

○ Confidentiality

○ Integrity

○ Availability

● Correctness

1
point

2.
What was the first buffer overflow attack?

● Morris Worm

○ SQL Slammer

○ Code Red

○ Love Bug

1
point

3.
The stack is memory for storing

● Local variables

○ Program code

○ Global variables

○

Dynamically linked libraries

1
point

4.
Why is it that the compiler does not know the absolute address of a local variable?

● As a stack-allocated variable, it could have different addresses depending on when its containing function is called

○ Programs are not allowed to reference memory using absolute addresses

○ Compiler writers are not very good at that sort of thing

○ The size of the address depends on the architecture the program will run on

1
point

5.
When does a buffer overflow occur, generally speaking?

○ when copying a buffer from the stack to the heap

● when a pointer is used to access memory not allocated to it

○ when the program notices a buffer has filled up, and so starts to reject requests

○ when writing to a pointer that has been freed

1
point

6.
How does a buffer overflow on the stack facilitate running attacker-injected code?

● By overwriting the return address to point to the location of that code

○ By changing the name of the running executable, stored on the stack

○ By writing directly to %eax the address of the code

○ By writing directly to the instruction pointer register the address of the code

1
point

7.

7.

What is a nop sled?

○ It is a method of removing zero bytes from shellcode

○ It is another name for a branch instruction at the end of sequence of nops

● It is a sequence of nops preceding injected shellcode, useful when the return address is unknown

○ It is an anonymous version of a mop sled

1
point

8.

The following program is vulnerable to a buffer overflow (assuming the absence of automated defenses like ASLR, DEP, etc., which we introduce in the next unit). What is the name of the buffer that can be overflowed?

```
 1  #include <stdio.h>
 2  #include <string.h>
 3
 4  #define S 100
 5  #define N 1000
 6
 7  int main(int argc, char *argv[]) {
 8    char out[S];
 9    char buf[N];
10    char msg[] = "Welcome to the argument echoing program\n";
11    int len = 0;
12    buf[0] = '\0';
13    printf(msg);
14    while (argc) {
15      sprintf(out, "argument %d is %s\n", argc-1, argv[argc-1]);
16      argc--;
17      strncat(buf,out,sizeof(buf)-len-1);
18      len = strlen(buf);
19    }
20    printf("%s",buf);
21    return 0;
22  }
```

● out

○ buf

○ msg

○ len

1
point

9.

Here is the same program as the previous question. What line of code can overflow the

Here is the same program as the previous question. What line of code can overflow the vulnerable buffer?

```c
#include <stdio.h>
#include <string.h>

#define S 100
#define N 1000

int main(int argc, char *argv[]) {
  char out[S];
  char buf[N];
  char msg[] = "Welcome to the argument echoing program\n";
  int len = 0;
  buf[0] = '\0';
  printf(msg);
  while (argc) {
    sprintf(out, "argument %d is %s\n", argc-1, argv[argc-1]);
    argc--;
    strncat(buf,out,sizeof(buf)-len-1);
    len = strlen(buf);
  }
  printf("%s",buf);
  return 0;
}
```

○ printf(msg)

○ printf("%s",buf);

○ strncat(buf,out,sizeof(buf)-len-1);

● sprintf(out, "argument %d is %s\n", argc-1, argv[argc-1]);

○ len = strlen(buf);

1 point

10.

Recall the vulnerable overflow from the previous two questions. We can change one line

Recall the vulnerable overflow from the previous two questions. We can change one line of code and make the buffer overrun go away. Which of the following one-line changes, on its own, will eliminate the vulnerability?

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

#define S 100
#define N 1000

int main(int argc,char *argv[]) {
  char out[S];
  char buf[N];
  char msg[] = "Welcome to the argument echoing program\n";
  int len = 0;
  buf[0] = '\0';
  printf(msg);
  while (argc) {
    sprintf(out,"argument %d is %s\n",argc-1,argv[argc-1]);
    argc--;
    strncat(buf,out,sizeof(buf)-len-1);
    len = strlen(buf);
  }
  printf("%s",buf);
  return 0;
}
```

○ change char msg[] = "Welcome to the argument echoing program\n" to char msg[42] = "Welcome to the argument echoing program\n"

● change sprintf(out, "argument %d is %s\n", argc-1, argv[argc-1])

to snprintf(out, S, "argument %d is %s\n", argc-1, argv[argc-1])

○ change printf("%s",buf) to printf(buf);

○ change printf(msg) to printf("%s",msg);

1 point

11.

Recall the vulnerable program from the previous few questions. Which of the following attacks do you think the program is susceptible to?

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

#define S 100
#define N 1000

int main(int argc, char *argv[]) {
  char out[S];
  char buf[N];
  char msg[] = "Welcome to the argument echoing program\n";
  int len = 0;
  buf[0] = '\0';
  printf(msg);
  while (argc) {
    sprintf(out, "argument %d is %s\n", argc-1, argv[argc-1]);
    argc--;
    strncat(buf,out,sizeof(buf)-len-1);
    len = strlen(buf);
  }
  printf("%s",buf);
  return 0;
}
```

○ code injection

○ data corruption

○ reading arbitrary addresses in memory

● all of the above

1
point

12.

Recall the program again.

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

#define S 100
#define N 1000

int main(int argc, char *argv[]) {
  char out[S];
  char buf[N];
  char msg[] = "Welcome to the argument echoing program\n";
  int len = 0;
  buf[0] = '\0';
  printf(msg);
  while (argc) {
    sprintf(out, "argument %d is %s\n", argc-1, argv[argc-1]);
    argc--;
    strncat(buf,out,sizeof(buf)-len-1);
    len = strlen(buf);
  }
  printf("%s",buf);
  return 0;
}
```

If we changed `printf("%s",buf)` to `printf(buf)` then the program would be vulnerable to what sort of attack?

○ heap overflow

● format string attack

○ use-after-free attack

○ all of the above

1
point

13.
Exploitation of the Heartbleed bug permits

○ a format string attack

○ a kind of code injection

○ overwriting cryptographic keys in memory

● a read outside bounds of a buffer

1
point

14.
Why is it that anti-virus scanners would not have found an exploitation of Heartbleed?

● Anti-virus scanners tend to look for viruses and other malicious

code, but Heartbleed exploits steal secrets without injecting any code

○ It's a vacuous question: Heartbleed only reads outside a buffer, so there is no possible exploit

○ Heartbleed attacks the anti-virus scanner itself

○ Heartbleed exploits are easily mutated so the files they leave behind do not appear unusual

1
point

15.

An integer overflow occurs when

○ an integer is used to access a buffer outside of the buffer's bounds

○ an integer is used as if it was a pointer

○ there is no more space to hold integers in the program

● an integer expression's result "wraps around"; instead of creating a very large number, a very small (or negative) number ends up getting created

☑ I, **Charvik Patel**, understand that submitting work that isn't my own may result in permanent failure of this course or deactivation of my Coursera account.
Learn more about Coursera's Honor Code

Submit Quiz