Feedback — week 1 quiz

Help

You submitted this quiz on **Sun 26 Oct 2014 2:07 PM PDT**. You got a score of **34.00** out of **37.00**. You can attempt again, if you'd like.

This timed quiz covers material from week 0 and week 1.

Question 1

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

Your Answer		Score	Explanation
● Correctness	✓	1.00	
○ Integrity			
○ Availability			
○ Confidentiality			
Total		1.00 / 1.00	

Question 2

What was the first buffer overflow attack?

Your Answer	Score	Explanation
○ SQL Slammer		

○ Code Red			
● Morris Worm	✓	1.00	
○ Love Bug			
Total		1.00 / 1.00	

Question 3

The stack is memory for storing

Your Answer		Score	Explanation
○ Dynamically linked libraries			
● Local variables	✓	3.00	
○ Program code			
○ Global variables			
Total		3.00 / 3.00	

Question 4

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

Your Answer		Score	Explanation
○ 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			
● As a stack-allocated variable, it could have different addresses depending on who called the function	✓	2.00	
○ Programs are not allowed to reference memory using absolute addresses			
Total		2.00 / 2.00	

Question 5

When does a buffer overflow occur, generally speaking?

Your Answer	Score	Explanation

○ when writing to a pointer that has been freed		
○ when copying a buffer from the stack to the heap		
○ when the program notices a buffer has filled up, and so starts to reject requests		
● when a pointer is used to access memory not allocated to it	✓	3.00
Total		3.00 / 3.00

Question 6

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

Your Answer		Score	Explanation
○ 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			
● By overwriting the return address to point to the location of that code	✓	3.00	
Total		3.00 / 3.00	

Question 7

What is a nop sled?

Your Answer		Score	Explanation
○ 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 ✓ 2.00

○ It is a method of removing zero bytes from shellcode

○ It is an anonymous version of a mop sled

Total 2.00 / 2.00

Question 8

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

```
#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;
}
```

Your Answer		Score	Explanation
○ len			
○ msg			
○ buf			
● out	✓	3.00	
Total		3.00 / 3.00	

Question 9

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

```
#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;
}
```

Your Answer		Score	Explanation
● sprintf(out, "argument %d is %s\n", argc-1, argv[argc-1]);	✓	3.00	This can overrun out, which is of limited size, e.g., by having a very large command-line argument

○ len = strlen(buf);

○ printf(msg)

○ printf("%s",buf);

○ strncat(buf,out,sizeof(buf)-len-1);

Total	3.00 / 3.00

Question 10

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? Check all correct answers.

```
#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;
}
```

Your Answer		Score	Explanation
☐ change printf("%s",buf) to printf(buf);	✓	1.00	This doesn't help; in fact, it introduces another possible bug.
☐ change while (argc) to while (!argc)	✓	1.00	The loop with the vulnerable code in it will never execute in this case, so that "fixes" the bug.
☑ 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]);	✓	1.00	This removes the overflow directly - snprintf limits the writes to the buffer to be no more than its length (N).
☑ change printf(msg) to printf("%s",msg);	✗	0.00	This has no real effect; msg never changes.
☑ change strncat(buf,out,sizeof(buf)-len-1); to	✗	0.00	This introduces a bug that wasn't there!

strlcat(buf,out,sizeof(buf));	
Total	3.00 / 5.00

Question 11

Recall the vulnerable program from the previous few questions. Which of the following attacks do you think the program is susceptible to (check all that apply)?

```
#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;
}
```

Your Answer		**Score**	**Explanation**
☑ reading arbitrary addresses in memory	✓	1.00	This is possible by injecting code that will read those addresses
☑ nontermination	✓	1.00	This is possible by injecting code that fails to terminate
☑ code injection	✓	1.00	This is possible by overwriting the return address to point to injected code.
☑ data corruption	✓	1.00	This is possible by overwriting data on overflow
Total		4.00 / 4.00	

Question 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?

Your Answer	Score	Explanation
○ heap overflow		

○ use-after-free attack			
● format string attack	✓	3.00	
○ all of the above			
Total		3.00 / 3.00	

Question 13

Exploitation of the Heartbleed bug permits

Your Answer		Score	Explanation
○ a kind of code injection			
○ a format string attack			
○ overwriting cryptographic keys in memory			
● a read outside bounds of a buffer	✓	1.00	
Total		1.00 / 1.00	

Question 14

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

Your Answer		Score	Explanation
○ Heartbleed attacks the anti-virus scanner itself			
○ Heartbleed exploits are easily mutated so the files they leave behind do not appear unusual			
○ 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	✖	0.00	reading outside a buffer cannot be used to inject code, but it can be used to steal secrets, so this statement is false
Total		0.00 / 1.00	

Question 15

An integer overflow occurs when

Your Answer		Score	Explanation
○ an integer is used to access a buffer outside of the buffer's bounds			
○ there is no more space to hold integers in the program			
○ an integer is used as if it was a pointer			
● an integer expression's result "wraps around" -- instead of creating a very large number, a very small number ends up getting created	✓	2.00	
Total		2.00 / 2.00	