Week 2 quiz

TOTAL POINTS 15

1. A program indexes a buffer after a pointer to that buffer has been used as a parameter to the `free()` function. This is

1 point

- (•) A violation of temporal memory safety
- ( ) An information flow violation
- ( ) A violation of spatial memory safety
- ( ) Correct behavior

- [ ] If the integer was used to perform pointer arithmetic
- [ ] If the integer is passed as an argument to `strncat`
- [ ] If the integer was passed as a parameter to `open()`
- [x] If the integer is used as the denominator in a division expression

3. Which of the following are true about a language that uses garbage collection or some other automatic means (e.g., reference counting) for memory management? (Select all that apply.)

1 point

- [ ] The language will not have type safety violations
- [x] The use of automatic memory management will provide a safety benefit, but typically at the cost of some performance

4. Consider the following code:

1 point

```
char *foo(char *buf) {
  char *x = buf+strlen(buf);
  char *y = buf;
  while (y != x) {
    if (*y == 'a')break;
    y++;
  }
  return y;
}
void bar() {
  char input[10] = "leonard";
  foo(input);
}
```

The definition of spatial safety models pointers as capabilities, which are triples *(p,b,e)* where *p* is the pointer, *b* is the base of the memory region the pointer is allowed to access, and *e* is the extent of that region. Assuming characters are 1 byte in size, what is a triple *(p,b,e)* for the variable `y` when it is returned at the end of the code?

- (•) (&input+4,&input,&input+10)
- ( ) (&input+4,&input,&input+7)
- ( ) (y,&input,buf)

5. Which of the following are true about a type-safe language? (Select all that apply.)

1 point

- [ ] The language is object-oriented.
- [ ] The language's types are static, i.e., checked by the compiler before running the program
- [x] The language may be used to enforce information flow security, depending on the type system
- [x] The language is sometimes memory safe, but not always

6. An engineer proposes that in addition to making the stack non-executable, your system should also make

1 point

- ( ) Ensure that memory is always deallocated
- ( ) Not make the program more secure, because attacker-controlled data cannot be stored in the heap
- (•) Make the program more secure by disallowing another location for an attacker to place executable code

7. What is the *best* choice of value for a stack canary, of the following options?

1 point

- ( ) The constant 0
- ( ) The constant 7
- (•) A random value
- ( ) A predictable value

attacker?

- ( ) The code in libc is better than code the attacker would write
- ( ) The injected code might have bugs
- ( ) There is no need to modify the application's executable code
- (•) There is no need to be able to execute (writable) data

9. In a return-oriented program (ROP), what is the role of the stack pointer?

1 point

- ( ) It's really no different than in a normal program
- (•) It's like the program counter in a normal program
- ( ) It's like the allocation pointer used by malloc()
- ( ) It's like the frame pointer in a normal program

flow graph because:

- ( ) The attacker is not interested in corrupting direct calls
- (•) CFI should be deployed on systems that ensure the code is immutable
- ( ) CFI should be deployed on systems that ensure the data is non-executable
- ( ) Programs that use CFI don't have direct calls

11. Recall that classic enforcement of CFI requires adding labels prior to branch targets, and adding code prior to the branch that checks the label to see if it's the one that is expected. Now consider the following program:

1 point

```
int cmp1(char *a, char *b) {
  return strcmp(a,b);
}

int cmp2(char *a, char *b) {
  return strcmp(b,a);
}
```

```
  int i;

  if(buf[0] == 'a') {
    p = cmp1;
  } else {
    p = cmp2;
  }

  printf("%d\n", p);
  strcpy(tmpbuff, buf);

  for(i = 0; i < sizeof(tmpbuff); i++) {
    if(tmpbuff[i] == 0) {
      break;
    } else {
      if(tmpbuff[i] > 97) {
        tmpbuff[i] -= 32;
      }
    }
  }

  return p(tmpbuff,buf);
}
```

To ensure that the instrumented program runs correctly when not being attacked, which of the following functions would have to be given the same label? Choose at least two, but no more functions than necessary.

- [x] cmp1
- [ ] cmpp
- [ ] strcmp
- [x] cmp2

12. In your review of a program, you discover the following function:

1 point

```
void afunction(char *buf) {
  static char BANNED_CHARACTERS[] = { '>', '<', '|', '*' };
  int i = strlen(buf);
  int l;

  for(l = 0; i < l; i++) {
    int j;
    int k = sizeof(BANNED_CHARACTERS) / sizeof(char);
    for(j = 0; j < k; j++) {
      if(buf[i] == BANNED_CHARACTERS[j])
        buf[i] = ' ';
    }
  }
```

How would you best describe what this function is doing?

- (•) Input sanitization by blacklisting
- ( ) Using a safe string library
- ( ) Input validation by whitelisting
- ( ) Spatial safety enforcement

13. A safe string library typically attempts to ensure which of the following?

1 point

- (•) That there is sufficient space in a source and/or target string to perform operations like concatenation, copying, etc.
- ( ) That wide (i.e., multibyte) character strings can be used where single-byte character strings are expected.
- ( ) That strings from the safe library can be freely passed to the standard string library functions, and

14. A project manager proposes a C coding standard where pointer variables must be assigned to `NULL` after being passed to `free()`. Doing so:

1 point

- ( ) Is a poor security decision, because NULL pointer dereferences could cause the program to crash
- (•) Stops writes to stale pointer values that might otherwise succeed and result in program compromise
- ( ) Prevents memory leaks, thus avoiding potential denial of service
- ( ) Helps code readability, but not security

15. A colleague proposes using a heap allocator that randomizes the addresses of allocated objects. This

1 point

- (•) Will make the program more secure, because attackers frequently rely on predicting the locations of heap-allocated objects in exploits
- ( ) Will increase performance by keeping the cache sparsely populated
- ( ) Will have no impact on security or performance
- ( ) Will make the program less secure, because the application will not be able to predict the locations of