CS 628A: Pop Quiz 1 Solutions

4 February 2019 (20 points/14 minutes)

This quiz reviews material taught in modules 0 and 1 of the course.

Name: _____
Roll number: _____
Moodle-registered email: _____

Instructions: The following are a mix of multiple-choice questions or fill-in-the-blanks. There is only one correct option for the multiple-choice questions. **Tick** or **circle** the most appropriate option among the answers provided.

Do not open the exam booklet until the instructor announces that the exam has started. You **must put down your pens/pencils** when the instructor says that time is up. Disobeying either of these commands will result in getting zero credit for this and all subsequent exams and pop quizzes.

Honor Pledge: Copy the academic integrity pledge below and sign your name under it.

I pledge my honor that I have faithfully followed the institute's academic integrity policy during this examination.

_____

_____

_____

_____

_____

_____

Name: Signature:

1. Recall that Ken Thompson proposed a trojan that would modify the login program to allow backdoor access to an attacker. Could we use formal verification to analyze the source code of the login program to detect this backdoor? (1 pt)
    a. Yes
    b. No: Thompson's trojan would not modify the source of the login program, but instead modifies the C compiler to emit the backdoor when compiling the login program.
    c. Maybe, because verification is undecideable the verifier may or may not terminate.

2. Who among the following **does not** get defrauded when an attacker perpetrates clickfraud on a website with ads from Google AdSense? (1 pt)
    a. Google
    b. The ad buyer
    c. The website operator: Both Google and the ad buyer are paying for fake clicks, so they lose money. The website operator gains money (and is likely the perpetrator)!

3. Suppose the following program is executed on an architecture where the stack grows *upwards* towards higher address (not downwards as on x86 or AMD64). (2 pts)

    Note: growing upwards means that an instruction like "push eax" stores the value of eax at the memory location pointed to by the stack pointer esp and then increments the stack pointer (esp) while instructions like "pop eax" and "ret" decrement the stack pointer.

```
int main(int argc, char* argv[])
{
  char buffer[8];
  gets(buffer);
  printf("hello, %s\n", buffer);
  return 0;
}
```

    Can this program be exploited via buffer overflow/stack smashing?
    (a) Yes, the return address of gets can be overwritten via buffer overflow (as opposed to the return address of main when the stack grows downwards).
    (b) No

4. The following is "shellcode" for the AMD64 platform: (2 pts)

```
push        rax
xor         rdx,  rdx
xor         rsi,  rsi
mov         rbx,  '/bin//sh'
push        rbx
push        rsp
pop         rdi
mov         al,   59
syscall
```

What is the purpose of the following three instructions: “push rbx; push rsp; pop rdi”?

push rbx pushes the string “/bin//sh” onto the stack (at this point rsp points to the string). The other two instructions are equivalent to mov rdi, rsp. rdi an argument to the syscall, and points to a string which contains the path to binary that is to be executed.

5. In the shellcode from question 4, the value ‘/bin//sh’ that is loaded into rbx is not NULL terminated. What does this imply? (2 pts)
   a. **The shellcode may or may not work correctly as NULL termination is required. The item on the stack after ‘/bin//sh’ is the value of rax due to the first push instruction. This code will work correctly if rax is 0, as that will NULL terminate the /bin//sh string (and also not mess with the syscall number.)**
   b. The shellcode always works correctly because NULL termination is **unnecessary**.
   c. The shellcode always works correctly **even though** NULL termination **is required**.

6. In the shell code from question 4, why do we use the instruction “xor rdx, rdx”? Why it is written in this particular way. (1 pt)

   We use it to clear (set to 0) rdx. It is written this way because the machine code for mov rdx, 0 contains a NULL character. Machine code for xor rdx, rdx does not contain a NULL character. Since buffer overflows exploit functions like strcpy which stop copying as soon as a NULL character is seen, it is desirable to have shell code without any NULL characters.

7. In the shellcode from question 4, which register contains the system call number? Which system call is being invoked? (1 pt)

   rax (al) contains the syscall number. The code is invoking execve.

8. What happen when the output of the following command: (2 pts)

```
$ python -c 'print "a"*9'
```

   is provided as input to the following C program. Assume compilation uses ProPolice (aka gcc -fstack-protector), the target architecture is x86_64/AMD64, the stack is 16-byte aligned and there is no padding between any two local variables.

```
int cmpchar(const void* d1, const void* d2)
{
      char p1 = *(const char*) d1;
      char p2 = *(const char*) d2;
      if (p1 < p2) return -1;
      if (p1 > p2) return  1;
      return 0;
}
```

```
int main(int argc, char* argv[])
{
      volatile int (*compar)(const void*, const void*) = cmpchar;
      char buffer[8];
      scanf("%s", buffer);
      qsort(buffer, strlen(buffer), sizeof(buffer[0]), compar);
      printf("%s\n", buffer);
      return 0;
}
```

a. **The program will crash with the message "stack smashing detected." ProPolice/stack-protector re-orders variables so that buffer overflows can't overwrite pointers, so 'compar' can't be overwritten by the overflow. Further, it puts a canary after the buffer, so any overflow (even by a single byte) is detected.**
b. The program will crash inside qsort.
c. The program will print 9 'a' characters.
d. Both options b and c are possible.

9. Python is a memory safe language, which among other things, means it is not possible to write past the end of array in Python. Does that mean control flow hijacking via buffer overflow vulnerabilities can never occur in Python programs? Explain your answer. (2 pts)

a. Correct, I never have to worry about buffer overflow vulnerabilities in Python programs.
b. No, it is still possible to have control flow hijacking via buffer overflow.

Unfortunately, the most commonly used Python interpreter (CPython) itself is a C program and many libraries used in Python are written in C. Both the interpreter and libraries may have buffer overflow vulnerabilities. One example is: https://bugs.python.org/issue20246.

10. Does the following code contain any vulnerability? If so, name the type of vulnerability. (2 pts)

```
int main(int argc, char* argv[]) {
  if (argc != 3) { fprintf(stderr, "syntax error.\n"); exit(1); }
  int n = atoi(argv[1]);
  int len = strlen(argv[2]);
  int sz = n * len;
  char* rptd = (char*) alloca(sz + 1 /* for '\0' at end */);
  if (rptd == NULL) { fprintf(stderr, "out of memory.\n"); exit(2); }
  for (char* p = rptd; p < rptd + sz; p += len) {
    strcpy(p, argv[2]);
  }
  printf("%s\n", rptd);
  return 0; }
```

a. No vulnerability
b. **Vulnerability type: This code has a lot of problems. First, there could be an integer overflow when we compute sz = n*len, and potentially another when we do sz+1. Second alloca does not return NULL when allocation fails, the behavior is undefined. This could lead to a buffer overflow.**

11. Which of the following is a secure way of choosing a value for a stack canary? (1 pt)
    a. By calling rand()
    b. By applying the SHA256 hashing algorithm on the return value of rand()
    c. By reading the canary from /dev/random
    d. Both b and c.

    Note that rand() does not provide cryptographically secure random numbers. (Note: Not cryptographically secure is a way of saying that the "random" numbers provided by rand can easily be predicted, which in our case would mean that the attacker could guess the canary.) Also note, applying SHA256 on a predictable "random" value does not make it any less predictable.

12. Which of the following protection schemes did not involve modifying the kernel? (1 pt)
    a. DEP
    b. ASLR
    c. ProPolice aka -fstack-protector: this just involves rearranging local variables on the stack and introducing a canary. It does not involve any kernel support at all.

13. Which of the following prompted the development of return-oriented programming (ROP) attacks? (1 pt)
    a. ProPolice aka -fstack-protector
    b. DEP: Because the stack was no longer executable, attackers developed ROP to execute code that was already part of the binary, rather than introducing new code on the stack.
    c. ASLR

14. Since C is not an object-oriented language, the C compiler does not generate any vtables. As a result, heap-based attacks cannot hijack control flow for C programs. (1 pt)
    a. True
    b. False: C has function pointers which may be stored on the heap. In fact, the first version of C++ was implemented as a translator to C in which vtables were turned into a function pointer array.

End of Exam